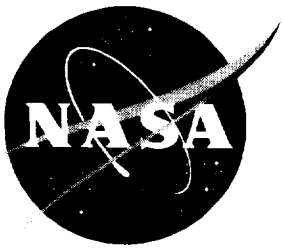


NASA Technical Paper 3452
Volume 2

HiRel: Hybrid Automated Reliability Predictor (HARP) Integrated Reliability Tool System (Version 7.0)

HARP Tutorial

Elizabeth Rothmann, Joanne Bechta Dugan, Kishor S. Trivedi, and Nitin Mittal
Duke University • Durham, North Carolina

Salvatore J. Bavuso
Langley Research Center • Hampton, Virginia

National Aeronautics and Space Administration
Langley Research Center • Hampton, Virginia 23681-0001

November 1994

This publication is available from the following sources:

NASA Center for AeroSpace Information
800 Elkridge Landing Road
Linthicum Heights, MD 21090-2934
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 487-4650

Contents

Chapter 1

Introduction

This document is a tutorial for the HARP software program, which is a member of the Hybrid Automated Reliability Predictor (HARP) integrated Reliability (HiRel) tool system for reliability/availability prediction (refs. 1 and 2). (See vol. 1 of this TP.)

HiRel offers a toolbox of integrated[1] reliability/availability programs that can be used to customize the user's application in a workstation or nonworkstation environment. HiRel consists of interactive graphical input/output programs and four reliability/availability modeling engines that provide analytical and simulative solutions to a wide host of highly reliable fault-tolerant system architectures and is also applicable to electronic systems in general. Three of the HiRel programs were developed by researchers at Duke University and at NASA Langley Research Center.

The tool system was designed to be compatible with most computing platforms and operating systems, and some programs have been beta tested within the aerospace community for over 8 years. Many examples of the system's use have been reported in the literature and at the HARP Workshop conducted at Duke University, July 10–11, 1990.

The wide range of applications of interest has caused HiRel to evolve into a family of independent programs that communicate with each other through files that each program generates. In this sense, HiRel offers a toolbox of integrated programs that can be executed to customize the user's application. Figure 1 shows the HiRel tool system. The core of this capability consists of the reliability/availability modeling engines, which are collectively called the Hybrid Automated Reliability Predictor (HARP).

The modeling engines are comprised of four self-contained executable software components: the original HARP program (described in vols. 1 and 2 of this TP), the Monte Carlo integrated HARP (MCI-HARP) (ref. 3), Phased Mission HARP (PM-HARP) (ref. 4), and X Window System HARP (XHARP) (ref. 5). In conjunction with the engine suite, are two input/output interactive graphical user-interface programs that provide a workstation environment for HiRel. These programs are called the Graphics Oriented (GO) program (described in vol. 3) and the HARP Output (HARPO) program (described in vol. 4). The base components of HiRel (GO, HARP, MCI-HARP, and HARPO) are available through NASA's software distribution facility, COSMIC,[2] or from the developers at Duke University.[3] The XHARP engine[4] is available from the university where it was developed. PM-HARP can be obtained from The Boeing Commercial Airplane Group.[5]

A number of examples are presented in this tutorial beginning with simple models and progressing to more complex ones to illustrate the HARP capability and to present more detail on the HARP modeling process. This tutorial only demonstrates the textual input/output HARP format. The developers were successful in retaining an identical textual input/output and file

[1] Integrated denotes the ability of HiRel software programs to communicate with each other in a common ASCII file format. These files are discussed in volume 1 of this Technical Paper.

[2] COSMIC, The University of Georgia, 382 East Broad St., Athens, GA 30602.

[3] Duke University, Dept. of Electrical Eng., Durham, NC, 27706 (Kishor S. Trivedi).

[4] Clemson University, Dept. of Computer Science, Clemson, SC 29734 (Robert Geist).

[5] The Boeing Commercial Airplane Group, Seattle, WA 98124 (Tilak C. Sharma).

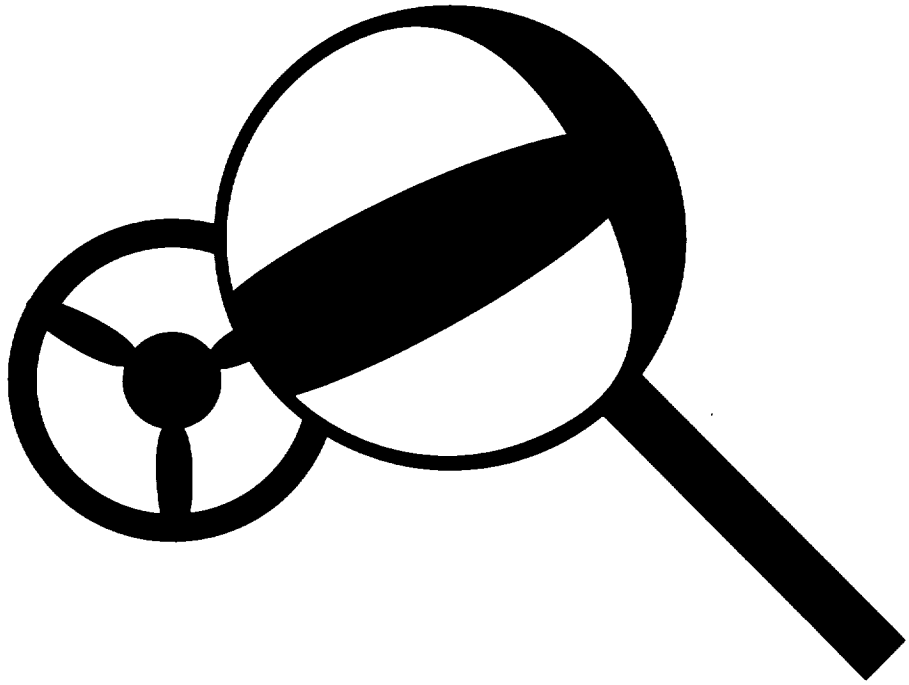


Figure 1. HiRel: GO, HARPO, and suite of reliability engines.

structure for all versions of HARP running on different computing platforms. This convenience was accomplished by implementing textual HARP in ANSI standard Fortran 77.

Graphical input/output capabilities are presented in volumes 3 and 4 of this Technical Paper. The graphical user interfaces (GUI), GO and HARPO, use the ANSI standard Graphical Kernel System (GKS) software to facilitate portability across several graphical display devices. Unlike the success achieved with textual HARP, the GUI's do not have identical appearances on the screens of different display devices associated with their different computing platforms. The difference in appearance is fortunately minimal and was dictated as such by the GKS installed on a particular computing platform (ref. 2).

Although some modeling concepts are explained in this document to illustrate the modeling process, the bulk of the theoretical concepts are presented in volume 1 of this Technical Paper and in several research papers cited in the reference section. The most comprehensive compilation of HiRel papers can be found in the proceedings of the HARP Workshop.

Combinatorial fault occurrence/repair models (FORM's) are initially presented in this volume. The single fault/error handling models (FEHM's) are presented next and are followed by the HARP multifault/error handling models applied to the near-coincident fault application. Appendix A provides file listings of worked examples from this tutorial, and appendix B provides additional examples with particular emphasis on the dynamic fault tree gates. Sequence dependency FORM's are also presented in appendix B.

Important concepts necessary to use HARP properly are presented in volume 1 of this Technical Paper, which should be read before any serious applications are undertaken with this capability. HiRel includes a number of software programs that are described in other volumes of this Technical Paper that may facilitate the user's productivity in using the HARP capability. Volumes 3 and 4 present the GO and the HARPO software programs, respectively. These documents describe the GUI for HiRel.

In the body of this document, a dialog is presented to illustrate the interaction between the user and the program. HARP commands are prefaced with the symbol $ or more commonly with no special prefix, and user responses within the sample sessions are identified with the symbol > preceding the response.

The GO, HARP, and HARPO HiRel software programs have been ported to many computing platforms and operating systems, which include Sun Microsystems, DEC VAX, IBM-compatibles 286, 386, and 486 PC's, Apollo, Alliant, Convex, Encore, Gould, Pyramid, and Berkley UNIX 4.3, AT&T UNIX 5.2, DEC VMS and Ultrix, and MS DOS, respectively.

The IBM-compatible PC 16-bit version requires a minimum of 512K of memory as well as a floating-point accelerator. Throughout the text, differences between the PC 16-bit version and the full version are noted. The PC 32-bit version running under DOS or OS/2 gives the full capability of the Sun or VAX versions.

The user is reminded that using HARP as a combinatorial fault tree solver is computationally inefficient, although convenient if the user is accustomed to using HARP. However, the fault tree is particularly useful when fault/error handling is included in the model or when sequence dependencies are modeled. Each model is no longer combinatorial.

Chapter 2

Creation of Files

This section presents an overview of the HARP program structure, execution flow, and the files it generates. Textural HARP executes on DEC VAX workstations under VMS, Sun Microsystem workstations under UNIX, and IBM-compatible 286, 386, and 486 PC's under MS DOS and OS/2. Textual HARP requires an ANSI standard Fortran 77 compiler and has been compiled with Lahey and Microsoft FORTRAN for PC's. It is compatible with a wide range of computing platforms because it was written in ANSI standard Fortran 77 for wide portability. HARP creates ASCII files, which are compatible with most computing platforms. For example, files created under the PC environment can be executed by a DEC VAX. In this way, a PC can be used as a workstation for input and output processing, and VAX can be used for large system computations. HARP has an interactive prompting input capability and is composed of three stand-alone programs: *tdrive*, *fiface*, and *harpeng*. (See fig. 2.) As the user successively executes the programs in this order, they create files that are required by downstream programs.

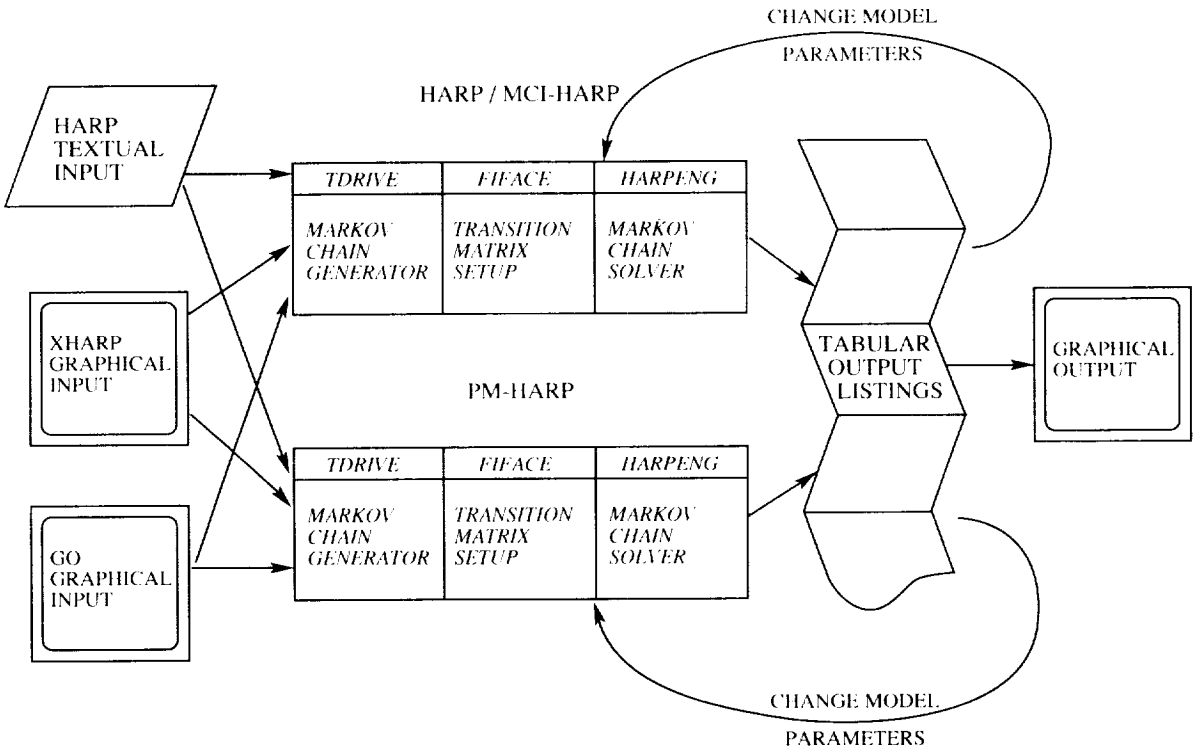


Figure 2. HARP execution and flow relationship to GO and HARPO.

The programs also accept user-generated or modified files created with a text editor. Thus, the user has the option to use the interactive input capability or simply input user-created files. The input to *tdrive* can also come from files generated by GO. The output of textual HARP are tabular structured files. These files can be used as input to HARPO, which allows the user to graphically display the tabular data in a wide variety of forms in an interactive mode. Thus,

as an overview, textual HARP is by analogy the central processing unit, GO is a graphical input to textual HARP that bypasses textual HARP's interactive input-prompting capability, and HARPO is the graphical output processor that reads textual HARP's tabular output files. (See vols. 3 and 4 of this TP.)

A brief description of the files created by the HARP programs is given here. A detailed presentation is given in volume 1 of this Technical Paper. The MODELNAME is specified by the user when the program *tdrive* is executed. The user should avoid special characters that are likely to interfere with the users' operating system; for example, a MODELNAME called * would be a bad choice.

2.1. Files Created by *tdrive*

If the user input is a fault tree, then *tdrive* creates the following files:

- MODELNAME.DIC A file that contains the name of each component in the model, its symbolic failure rate, and any fault/error handling information. This file is called the dictionary file.
- MODELNAME.FTR An interim file created by the program or the fault tree file from the HARP graphical input program.
- MODELNAME.INT The fault tree is converted to a Markov chain. This file contains the states and state transitions of the Markov chain after conversion.
- MODELNAME.TXT This file contains the textual fault tree description given by the user.

If the user input is a Markov chain, then *tdrive* creates the following files:

- MODELNAME.DIC A file that contains the name of each component in the model, its symbolic failure rate, and any fault/error handling information. This file is called the dictionary file. This file is *optional* for Markov chain input but imperative for fault tree input.
- MODELNAME.INT This file contains the states and state transitions of the Markov chain as input by the user. An important point for the Markov chain input: the first state listed in the MODELNAME.INT file must be the initial state of the system. That is, if the first line reads STATE1 STATE2 3*LAMBDA, then STATE1 is assumed to be the initial state of the Markov chain.

2.2. Files Created by *fiface*

The following files are created by *fiface*:

- MODELNAME.ALL This file contains the all-inclusive next faults rates if necessary; otherwise, it is empty.
- MODELNAME.MAT This file contains the sparse matrix format of the Markov chain. Row and column values of nonzero entries are listed in ascending order.
- MODELNAME.SAM This file contains the same-type next faults rates if necessary; otherwise, it is empty.
- MODELNAME.SYM This file contains symbol table information for the HARP engine program.
- MODELNAME.USR This file contains the user-defined next faults rates if necessary; otherwise, it is empty.

2.3. Files Created by *harpeng*

The following files are created by *harpeng*:

- MODELNAME.INP—This file contains the user input values for symbolic failure and repair rates. If desired, this file can be used for future runs (called an *echo file* by the *harpeng* program.
- MODELNAME.PT*—This file contains the unreliability values. It can be used if a plot program is available. (The symbol * is an integer from 1 to 9.)
- MODELNAME.RS*—This file contains the results of the program execution. The file lists the values given to symbolic failure or repair rates, solution values for coverage models, failure state probabilities (if input is a Markov chain, it can have some active state probabilities if requested), and unreliability and reliability values and bounds information. (The symbol * is an integer from 1 to 9.)

Chapter 3

Fault Occurrence/Repair Model (FORM)

This chapter addresses the construction and interpretation of FORM's for fault trees and Markov chains. We begin with a model of a simple system consisting of three processors. Only one processor is needed for the system to remain operational. For now, fault/error handling mechanisms are disregarded. When a processor fails, it is simply discarded and no recovery or repair is attempted.

3.1. Three-Processor System Fault Tree

The fault tree input is demonstrated first. Because fault/error handling is ignored, the FEHM model type none is used when entering the dictionary information. After entering the dictionary information, the structure of the model is entered as portrayed in the fault tree of figure 3. When using HARP with textual input, the user normally first sketches the system fault tree and labels it as shown in figure 3. Each member of the fault tree is labeled with a unique node number. During the input dialog, the user is asked to identify the connection of the fault tree members by specifying the node numbers. In this example, the node numbers happen to correspond to the basic event component ID numbers, also called the type numbers. Although the node numbers must be unique, the basic event component ID's are not required to be unique, that is, all the basic events can be the same type, say 1. A component ID is a positive integer that points to a dictionary description of the specified basic event. As the user inputs the NAME for the component ID, as prompted by HARP, the dictionary file is automatically created in ASCII format and can be viewed after the software program *tdrive* completes execution. The dictionary file contains the component name, the symbolic failure rate[6] for that component, and any specified FEHM. During execution of *harpeng*, the user is asked to specify the failure distribution and its numerical values for each component. During initial model input requested by *tdrive*, the user is asked to identify the component ID for each basic event node and to specify a replication factor, a positive integer. The use and significance of the replication factor are demonstrated in section 3.2.

Upon completion of the first program *tdrive*, the model has been converted to a Markov chain, although this process is transparent to the user. An ASCII file containing the Markov chain is created and identified as MODELNAME.INT. The corresponding Markov chain is shown in figure 4. The state 1,1,1 represents the system with each of the processors operating. With rate $\lambda 3$, the third processor fails, and the system enters the state with only the first two processors running, that is, state 1,1,0. Likewise, with rate $\lambda 2$, the second processor fails, and the system enters state 1,0,1. With rate $\lambda 1$, the first processor fails, and the system enters state 0,1,1. Now from this state, one of two events can occur: either the second processor can fail or the third processor can fail. The first event leaves the system in state 0,1,0 and the second event leaves the system in state 0,0,1. Analogous transitions emanate from states 1,1,0 and 1,0,1. Once there are two failures, (i.e., states 1,0,0; 0,1,0; and 0,0,1) the next failure crashes the system. From these states, F1 is entered upon failure of the first processor, F2 when the second processor fails and

[6] A better term is failure distribution; failure rate is used instead to simplify the input.

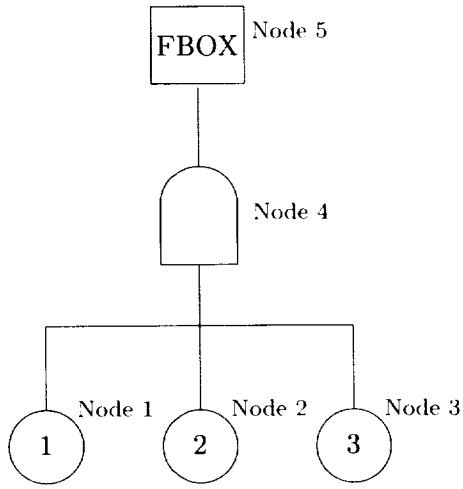


Figure 3. Three-processor fault tree representation.

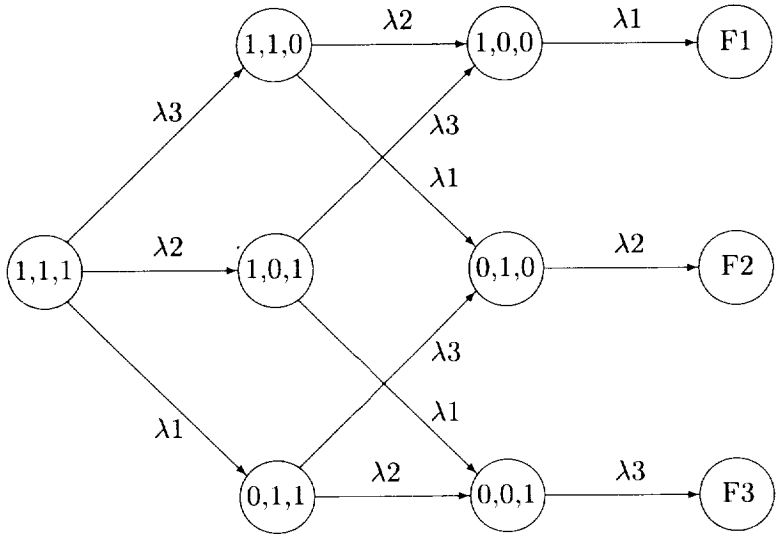


Figure 4. Markov chain generated from fault tree of figure 3.

F3 when the third processor fails. The program *fiface* is executed to create the sparse matrix[7] data structure format needed by the HARP engine. Finally, the engine itself is executed.

The three program dialogs for the example shown in figure 4 are presented in the following sections. In the dialogs, a program request has no special prefix, and a user response is preceded by the symbol >.

3.1.1. *tdrive* Dialog for Input of Fault Tree

In the following dialog, the program *tdrive* creates the file 3PFT1.INT that contains the Markov chain generated from the input fault tree. The data in the 3PFT1.INT file are always printed in ascending row-wise order with state names being positive integers. This output is called SORTED output. 3PFT1.DIC lists the dictionary information. Both of these files are in

[7] Matrix is $A(t)$ as described in volume 1 of this Technical Paper. A matrix is sparse when most of its entries are zero.

appendix A as is 3PFT1.TXT, which contains the fault tree input information. The dialog is as follows:

```
$  tdrive

                    HARP---Version 7.0, February 1993
                NASA Langley Research Center/Duke University
                              Program Tdrive
          Defaults are Invoked by "CR", Inputs are Case Insensitive
                          Question?  ( "?" or "help" )
                    FAULT TREE (F) or Markov Chain (M)?
>  f
Modelname?
>  3pft1          * Must be a legal filename without extension, .e.g., *
                  * 8 max. characters on a PC]                          *
 NAME for component ID    1.  Enter  "/d" or "done" if finished.
>  processor1             * Avoid using special characters              *
Symbolic failure rate?    * Numerical values are requested in fiface    *
                          * and harpeng. Avoid using special characters *
                          * such as $, &, etc.                          *
>  lambda1
Component FEHM?
>  none
Default Selected:  FEHM Model set to "NONE"
Continue => Y   Reenter => N
>  y
 NAME for component ID    2.  Enter  "/d" or "done" if finished.
>  processor2
Symbolic failure rate?
>  lambda2
Component FEHM?
>  none
Default Selected:  FEHM Model set to "NONE"
Continue => Y   Reenter => N
>  y
 NAME for component ID    3.  Enter  "/d" or "done" if finished.
>  processor3
Symbolic failure rate?
```

```
>  lambda3
Component FEHM?
>  none
Default Selected:  FEHM Model set to "NONE"
Continue => Y   Reenter => N
>  y
 NAME for component ID    4.  Enter  "/d" or "done" if finished.
>  done
                          Fault Tree Description.
Enter "/d" or done" for gate/box entry, ? for dictionary,
 or "/X" to correct input error.
Basic event node  1:
Component ID?          * HARP associates an integer with each component     *
                       * name shown in the modelname.dic file to simplify   *
                       * the code and to simplify basic event specification *
>  1
Replication factor?    * When basic events have the same failure rate symbol, *
                       * the specification of a replication factor greatly    *
                       * simplifies the HARP created Markov chain model       *
>  1
Summary:  Basic event node   1:   1 of component  1
Continue => Y  Reenter => N, (Default = Y)
>  y
Enter "/d" or done" for gate/box entry, ? for dictionary,
 or "/X" to correct input error.
Basic event node  2:
Component ID?
>  2
Replication factor?
>  1
Summary:  Basic event node   2:   1 of component  2
Continue => Y  Reenter => N, (Default = Y)
>  y
Enter "/d" or done" for gate/box entry, ? for dictionary,
 or "/X" to correct input error.
```

```
 Basic event node  3:
 Component ID?
 >  3
 Replication factor?
 >  1
 Summary:  Basic event node   3:   1 of component  3
 Continue => Y  Reenter => N, (Default = Y)
 >  y
 Enter "/d" or done" for gate/box entry, ? for dictionary,
  or "/X" to correct input error.
 Basic event node  4:
 Component ID?
 >  done
 Enter "/X" to correct input error, ? for help.
    Node   4: Gate or Box or Fbox (Enter "FBOX" as last node)
 Enter gate type:
 >  and
 Enter number of incoming arcs:     * Specify the actual number of arcs, not *
                                    * the replicated number, i.e., for       *
                                    * Replication factor =3, specify one arc *
 >  3
  Enter ID number of source node for arc     1:    * Its a good idea to first  *
                                                   * draw a sketch of the tree *
                                                   * labeling the nodes
                                                   * numerically,see fig. 12,  *
                                                   * Users Guide, vol.1.       *
 >  1
  Enter ID number of source node for arc     2:
 >  2
  Enter ID number of source node for arc     3:
 >  3
  SUMMARY:  Node   4: TYPE AND          ,   3 INPUTS:   1  2  3
 Continue => Y  Reenter => N, (Default = Y)
 >  y
 Enter "/X" to correct input error, ? for help.
```

```
   Node   5: Gate or Box or Fbox (Enter "FBOX" as last node)
Enter gate type:
>  fbox
 Enter ID number of source node for arc    1:
>  4
Summary: FBOX node  5:  INPUT:    4
Continue => Y  Reenter => N, (Default = Y)
>  y
TRUNCATE the model after how many failures?   * State truncation bounds, *
                                              * see sec. 3.4.2, vol. 1   *
>  0
Default selected: no truncation.
Include state tuples as comments in .INT file?   * tdrive will convert the    *
                                                 * fault tree into an equi-   *
                                                 * valent Markov chain. State *
                                                 * tuples identify each state *
>  n
Default selected: No state tuple notation.
FT2MC: Converting fault tree to Markov chain . . .
FT2MC: Successful completion
           8 internal Markov chain states generated
           7 unique nonfailure states                * 7 merged operational *
                                                     * states were formed   *
           3 failure states generated for HARP engine
Model information in file:      3PFT1.INT
Dictionary information in file: 3PFT1.DIC
```

3.1.2. *fiface* Dialog for Fault Tree Model

The next step is to run *fiface*. Its purpose is twofold: (1) *fiface* puts the Markov chain into the correct format needed by the HARP engine (a sparse matrix format with entries in column order), and (2) *fiface* adds any necessary coverage information. In this example, only the first task is applicable because no FEHM's were specified. The output files of *fiface* are 3PFT1.MAT, which contains the matrix and 3PFT1.SYM, which contains symbolic information. These files are also in appendix A. The dialog is as follows:

```
$  fiface
                    HARP---Version 7.0, February 1993
                              Program FIFACE
```

```
Modelname?
>  3pft1
Matrix and symbol table information in: 3PFT1.MAT  * fiface created the      *
                                                   * transition matrix ,     *
                                                   * see Users Guide, vol.1, *
                                                   * section 1.3             *
```

3.1.3. *harpeng* Dialog for Fault Tree Model Solution

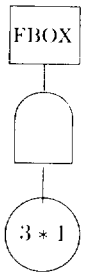
Next, we run the engine *harpeng* to obtain the solution to the problem. The results are stored in file 3PFT1.RS1 and are given in appendix A. The dialog is as follows:

```
$  harpeng
                ----------------- HARP -----------------------
                - The Hybrid Automated Reliability Predictor -
                ----------- Release Version 7.0 --------------
                ------------- February 1993 ------------------
 Use an echo file from a previous run as the input file?  y/n ?
>  no                        * An echo file is automatically written by     *
                             * the execution of harpeng which contains      *
                             * all the input data.  This file may be         *
                             * altered with a text editor for multiple       *
                             * executions of harpeng when the model          *
                             * configuration is unchanged                    *
 Modelname ?
>  3pft1
 Output files:
     3PFT1.RS1       --  Reliability and state probabilities
     3PFT1.PT1       --  Graphics information
                              ----- WORKING -----
     3PFT1.INP       --  Input file or echo of input
 Declare meaning for symbol  LAMBDA1       ( "?" or "help" )
>  1
 For constant failure rate: LAMBDA1
 Nominal value?
>  .001
 (+/-) Variation? (Must be less than nominal.          "?" will allow reentry. )
                                * Variation not asked in PC 16-bit HARP version *
```

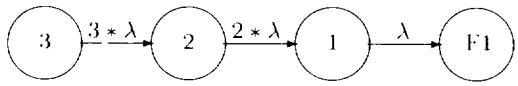
```
> 0
 Declare meaning for symbol  LAMBDA2      ( "?" or "help" )
> 1
 For constant failure rate: LAMBDA2
 Nominal value?
> .001
 (+/-) Variation? (Must be less than nominal.          "?" will allow reentry. )
                          * Variation not asked in PC 16-bit HARP version *
> 0
 Declare meaning for symbol  LAMBDA3      ( "?" or "help" )
> 1
 For constant failure rate: LAMBDA3
 Nominal value?
> .001
 (+/-) Variation? (Must be less than nominal.          "?" will allow reentry. )
                          * Variation not asked in PC 16-bit HARP version *
> 0
 Redefine symbol(s) meanings or their values, or correct an error (y/n)?
> n
 Mission time? (Hours):
> 10
 Mission time reporting interval? (Hours):
> 10
 Compute Parametric Bounds using SIMPLE Model? (y/n) ?  no
                          * Bounds disallowed in PC 16-bit HARP version *
Calculating State Probabilities...
        0 Reports from the GERK ODE solver.     * The non-stiff ordinary      *
                                                * differential solver reports *
                                                * any unusual long solutions  *
 Please select:
 1: Scroll through the result file?
 2: Solve same model with new mission time or near-coincident fault rates, etc.?
 3: Redefine symbol(s) meaning(s) and re-run model?
 4: Exit the program?
> 4
```

3.2. Variation of Three-Processor System Fault Tree

In the previous example, $\lambda 1 = \lambda 2 = \lambda 3$ indicates that the processors are statistically identical components. Because we are not concerned with which processor fails (merely with the fact that a fault has occurred), we can lump the processors into a single basic event as demonstrated in figure 5(a). The notation $3 * 1$ designates three replications of dictionary component type 1 (the processors in this case). The *tdrive* program converts the fault tree to the Markov chain shown in figure 5(b). Notice that the number of states in the Markov chain is reduced from 10 in the previous example to 4. State 3 represents the fully operational system. With failure rate $3 * \lambda$ (the coefficient 3 is the number of processors available), the system makes a transition to state 2 where only two processors are available. Likewise, with rate $2 * \lambda$ the system goes to state 1 with only one processor and finally, the failure of the remaining processor with rate λ brings the system down.



(a) Fault tree with replicated events.



(b) Corresponding Markov chain.

Figure 5. Three-processor system.

Figure 6 shows which states are being merged in the Markov chain generated from the fault tree of the previous example.

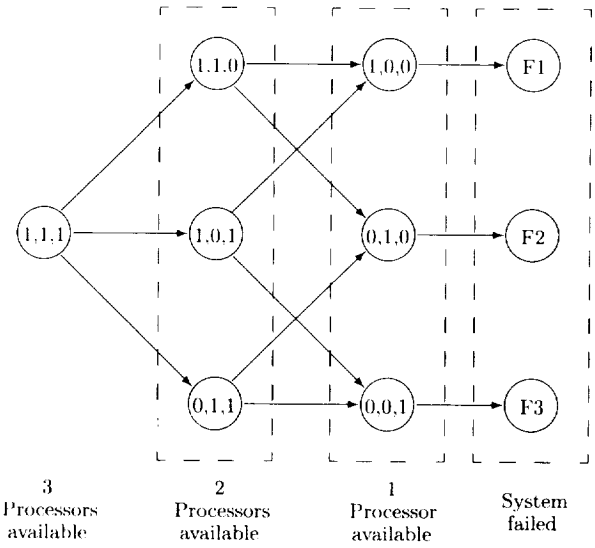


Figure 6. Merging of three-processor Markov chain.

The three program dialogs for the merged fault tree example are presented in the following sections. As in the previous example, and for all subsequent examples, the output files are listed in appendix A.

3.2.1. *tdrive* Dialog for Input of Merged Fault Tree

```
$  tdrive
                        HARP---Version 7.0, February 1993
                    NASA Langley Research Center/Duke University
                                Program Tdrive
            Defaults are Invoked by "CR", Inputs are Case Insensitive
                        Question?  ( "?" or "help" )
                    FAULT TREE (F) or Markov Chain (M)?
>  f
Modelname?
>  3pft2
 NAME for component ID     1.  Enter  "/d" or "done" if finished.
>  processor
Symbolic failure rate?
>  lambda
Component FEHM?
>  none
Default Selected:  FEHM Model set to "NONE"
Continue => Y   Reenter => N
>  y
 NAME for component ID     2.  Enter  "/d" or "done" if finished.
>  done
                              Fault Tree Description.
Enter "/d" or done" for gate/box entry, ? for dictionary,
 or "/X" to correct input error.
Basic event node  1:
Component ID?
>  1
Replication factor?
>  3
Summary:  Basic event node   1:   3 of component  1
Continue => Y  Reenter => N, (Default = Y)
```

```
> y
Enter "/d" or done" for gate/box entry, ? for dictionary,
 or "/X" to correct input error.
Basic event node  2:
Component ID?
> done
Enter "/X" to correct input error, ? for help.
   Node   2: Gate or Box or Fbox (Enter "FBOX" as last node)
Enter gate type:
> and
Enter number of incoming arcs:
> 1
 Enter ID number of source node for arc    1:
> 1
 SUMMARY:  Node   2: TYPE AND          ,   1 INPUTS:   1
Continue => Y  Reenter => N, (Default = Y)
> y
Enter "/X" to correct input error, ? for help.
   Node   3: Gate or Box or Fbox (Enter "FBOX" as last node)
Enter gate type:
> fbox
 Enter ID number of source node for arc    1:
> 2
Summary: FBOX node  3:  INPUT:    2
Continue => Y  Reenter => N, (Default = Y)
> y
TRUNCATE the model after how many failures?
> 0
Default selected: no truncation.
Include state tuples as comments in .INT file?
> n
Default selected: No state tuple notation.
FT2MC: Converting fault tree to Markov chain . . .
FT2MC: Successful completion
            4 internal Markov chain states generated
            3 unique nonfailure states
            1 failure states generated for HARP engine
Model information in file:      3PFT2.INT
Dictionary information in file: 3PFT2.DIC
```

3.2.2. *fiface* Dialog for Merged Fault Tree Model

```
$  fiface
                        HARP---Version 7.0, February 1993
                                  Program FIFACE
  Modelname?
>  3pft2
Matrix and symbol table information in: 3PFT2.MAT
```

3.2.3. *harpeng* Dialog for Merged Fault Tree Solution

After executing *harpeng*, the user should compare the results file 3PFT2.RS1 with the previous example 3PFT1.RS1. As expected, the unreliability values for each are identical because $\lambda 1 = \lambda 2 = \lambda 3$. By merging the states, the user can greatly reduce the size of the corresponding Markov chain and make analysis much faster (and if the model is large, can even make an otherwise intractable solution possible).

```
$  harpeng
                  ----------------- HARP ----------------------
                  - The Hybrid Automated Reliability Predictor -
                  ------------ Release Version 7.0 -------------
                  -------------- February 1993 -----------------
 Use an echo file from a previous run as the input file?  y/n ?
>  no
 Modelname ?
>  3pft2
 Output files:
     3PFT2.RS1       --  Reliability and state probabilities
     3PFT2.PT1       --  Graphics information
                              ----- WORKING -----
     3PFT2.INP       --  Input file or echo of input
 Declare meaning for symbol  LAMBDA        ( "?" or "help" )
>  1
 For constant failure rate: LAMBDA
 Nominal value?
>  .001
 (+/-) Variation? (Must be less than nominal.          "?" will allow reentry. )
>  0
 Redefine symbol(s) meanings or their values, or correct an error (y/n)?
>  n
```

```
 Mission time? (Hours):
>  10
 Mission time reporting interval? (Hours):
>  10
 Compute Parametric Bounds using SIMPLE Model? (y/n) ?  n
Calculating State Probabilities...
        0 Reports from the GERK ODE solver.
 Please select:
 1: Scroll through the result file?
 2: Solve same model with new mission time or near-coincident fault rates, etc.?
 3: Redefine symbol(s) meaning(s) and re-run model?
 4: Exit the program?
>  4
```

3.3. Three-Processor System Input as a Markov Chain

We now input the three-processor system as the Markov chain of figure 7. The description of the chain is given in the previous example. In program *tdrive*, the dictionary need not be entered for Markov chains that do not have repair nor fault handling. (The dictionary *must* be entered for a fault tree, a Markov chain with repair, or a Markov chain with coverage.) The program *fiface* is executed to create the sparse matrix data structure format needed by the HARP engine. The engine, as before, is run to solve the model. Output files are given in appendix A.

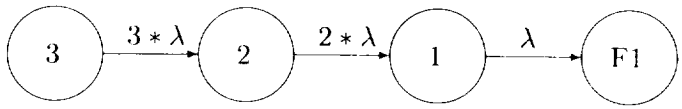


Figure 7. Three-processor system input as a Markov chain.

For the Markov chain input, the user is asked some different questions than for the fault tree input. When a fault tree is converted to a Markov chain in *tdrive*, the output is always printed in ascending row-wise order with state names being positive integers. This output is called SORTED output and is printed in the MODELNAME.INT file. For Markov chain input, the user has the choice of entering data in a SORTED or UNSORTED manner. If the entries are UNSORTED, it means one of two things; either the state names are symbolic (STATE1, F1, 3P, etc.) or the entries are not in row-wise ascending integer order. Additionally, an instruction is posted that tells the user that the first state listed must be the initial state of the system. If this is not the case, the solution results can be incorrect. Next, the user is asked if the model is to be solved AS IS. The AS IS option means that there are no fault/error handling models for any of the components and only the FORM is to be solved.

3.3.1. *tdrive* Dialog for Input of Three-Processor System

```
$  tdrive
                        HARP---Version 7.0, February 1993
                    NASA Langley Research Center/Duke University
                                Program Tdrive
            Defaults are Invoked by "CR", Inputs are Case Insensitive
                            Question?  ( "?" or "help" )
                        FAULT TREE (F) or Markov Chain (M)?
>  mc
Modelname?
>  3pmc
 Will any FEHMs be used? (y/n) ?  n
Will state names be in row-wise order listed as ascending integers
beginning with 1? (i.e., no symbolic input for state names).
    (y/n  default = n ) ?  n
The first state entered must be the initial state,i.e., for the
line -- S1 S2 RATE  -- "S1" is the initial state.
Begin Markov chain entry with "read filename", or simply list
the transitions using the format:  S1  S2  Rate_expression
(Enter "/d" or "done", "?" or "help")
Begin:
 >  3  2  3*lambda
 >  2  1  2*lambda
 >  1  F1 lambda
 >  done
Model information in file:       3PMC.INT
Dictionary information in file: 3PMC.DIC
```

3.3.2. *fiface* Dialog for Three-Processor System

Now run the *fiface* program as before. This time the user is asked whether the model has repair (for this example, there is no repair). In addition, the user is asked whether any active state probabilities are desired. Thus, the user can obtain the probabilities for any state, not just the failure states (as in the fault tree, SORTED, input). After running *fiface*, use a text editor to compare the contents of the 3PFT2.MAT file from this run with that of the previous example (3PFT2.MAT from section 3.2). They are the same. It serves to reason that the results files (3PFT2.RS1) are also the same. Compare the results files in appendix A. The *harpeng* run is not listed here because it is identical to the previous run. The dialog is as follows:

```
$  fiface

                    HARP---Version 7.0, February 1993

                              Program FIFACE

Modelname?

>  3pmc

Model is to be solved "as is".

Matrix and symbol table information in: 3PMC.MAT

 Does this model have repair?  - y/n:

>  n

    PLEASE NOTE: THE  FIRST STATE IN THE .INT

    FILE IS CONSIDERED THE INITIAL STATE OF THE MODEL.

 Do you want to see the state probabilities for any active states?

 This information is automatically printed for any failure states. (y/n)?

>  n
```

Chapter 4

Fault/Error Handling Model (FEHM)

4.1. Full Model

Let us expand our three-processor example. (See fig. 7.) Each processor in states 3, 2, and 1 still fails at a constant rate λ. However, upon processor failure, the system enters the Fault Active state. (See fig. 8.) In this state, the system attempts to detect the fault with a constant detection rate δ. If the fault is detected with probability p, the faulty processor is removed and the system enters the state with one fewer processor. Otherwise, if the fault goes undetected, it propagates through the system causing system failure with probability $1 - p$. This single-point failure state is recognized as state FSPF, which is a failure due to a single-point fault. If all faults are detected, we eventually exhaust our supply of processors entering the failure state F1.

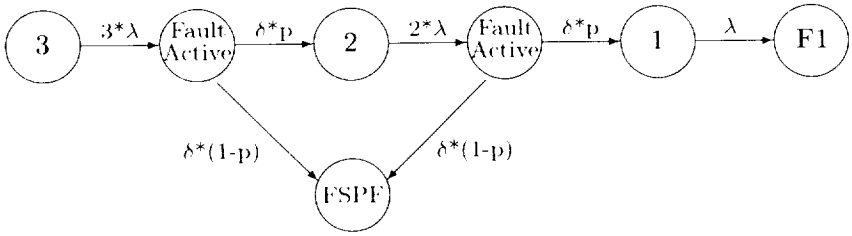


Figure 8. Three-processor system with active faults.

Next, we generalize and replace the Fault Active state shown in figure 8 with a box, perhaps containing many states, as shown in figure 9. Each box contains the "fast" transitions of fault recovery and hence is referred to as the FEHM. The FEHM captures in a few parameters the sequence of events that occur within the system once a fault occurs. Its general structure is a single-entry (up to) four-exit model, which is entered when a fault occurs. The exits represent possible outcomes of the attempted system recovery. As demonstrated in figure 9, the FEHM can be inserted only between operational states.

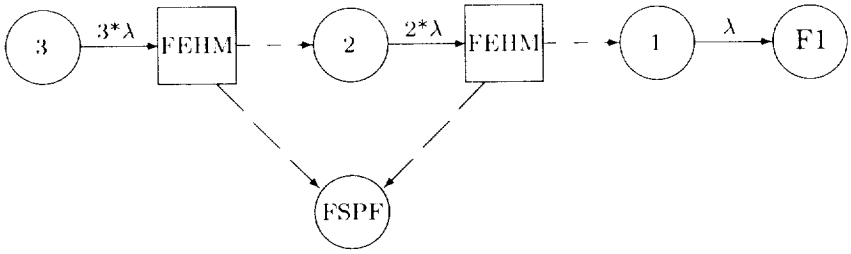


Figure 9. Three-processor system with FEHM's inserted.

In general, what is inside the box may or may not be a single Markovian state, which is the Fault Active state shown in figure 8. It can be as simple or as complex as the user wants. For the moment, it does not matter what is in this box. What is important is that we analyze the FEHM to determine the probability of successful permanent coverage—that is, detecting the fault and reconfiguring with one fewer processor. Accordingly, with the complementary probability, the

system fails. The path taken in the case of a successful reconfiguration is the C exit from the FEHM and the path to system failure is the S exit. For this example, the contents of the FEHM are represented by the Fault Active state of figure 8. The structure of the FEHM for this example is shown in figure 10.

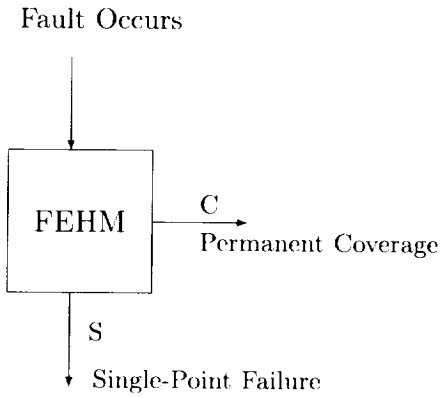


Figure 10. Partial structure of HARP FEHM.

As shown in figure 11, the FEHM box has been reduced to a branch point. The parameter c (in this case, $c = p$) represents the probability of successful detection and reconfiguration. The complementary probability parameter leading to state FSPF is denoted by an s (in this case, $s = 1 - p$). The overall model that is solved to predict the reliability of the system is shown in figure 12.

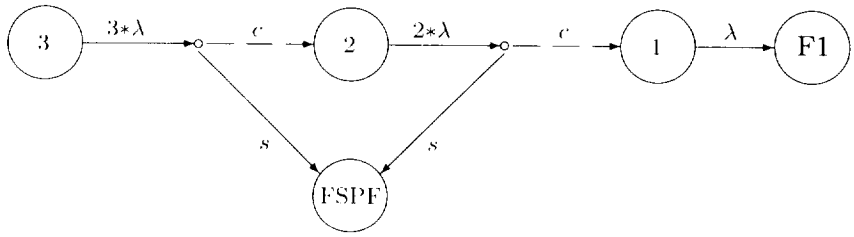


Figure 11. Replacing FEHM's by a branch point.

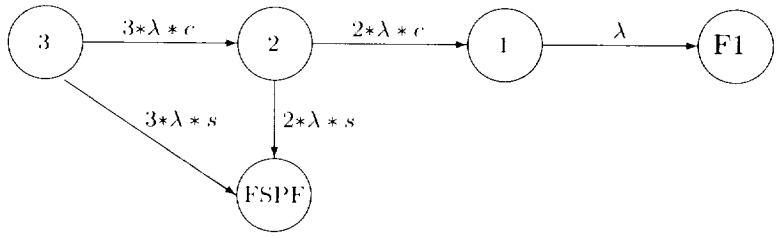


Figure 12. Instantaneous jump model of three-processor system.

4.2. Development of Instantaneous Jump Model

The reduction from the full model of figure 8 to the instantaneous jump model of figure 12 is the general procedure HARP uses to solve large and stiff models. Models, such as figure 8, with many orders of magnitude between the slowest and fastest rates are called stiff systems.

We separate the system along temporal lines (with respect to occurrence times) according to the relative magnitude of the state transition rates. The states representing failures (slow transitions) are grouped into the FORM and the fast recovery states are grouped into the FEHM (also referred to as the coverage model). This is the concept of behavioral decomposition (refs. 6 and 7). The FEHM is solved in isolation, reduced to a branch point, and inserted into the FORM, as shown in the example.

Behavioral decomposition is used not only for model solution but also for model specification. The user enters the FORM and FEHM separately and thus is shielded from specifying a huge overall model. Note that the combined model, like that of figure 8, which is both stiff and potentially large, is *never* constructed by the user nor generated by HARP. The solution is designed to use a good decomposition approximation so that a small nonstiff model is solved rather than a large stiff model. This largeness avoidance technique is the basis of HARP.

The user should ensure that adequate separation (at least two orders of magnitude) occurs between the parameters in the FORM and FEHM models. Otherwise, the results produced by HARP can produce an unacceptable conservative result. In the event such a condition results, HARP issues a warning message to that effect. The degree of acceptable conservative error is a function of the fidelity of the model, the accuracy of the input data (which is typically in error by at least one order of magnitude), user requirements, and other less important factors. Engineering judgment is the prime consideration when any modeling data are accepted.

Chapter 5

Modeling Permanent Faults

Assume that the boxes in figure 9 now represent a subset of the CARE III single fault model (ref. 8) to demonstrate the idea of permanent faults. A fault is permanent if its faulty manifestation persists for a long time. The time period is relative to the criticality of the application, so for a flight control application, a long time would be on the order of tenths of a second. For a system such as that shown in figure 9, the best action upon detecting a processor with a permanent fault is to discard the processor. Thus, the system survives and functions with one less processor. A portion of the CARE III model is shown in figure 13.

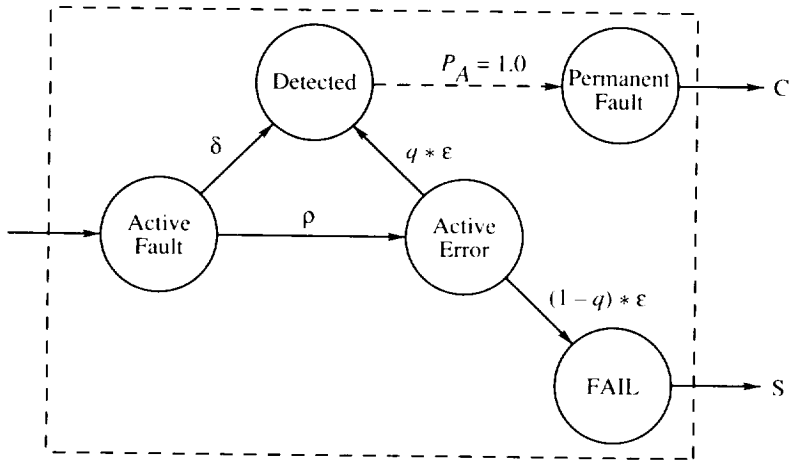


Figure 13. Portion of CARE III single fault model.

The fault is detected with constant rate δ. Once detected, the system removes the faulty unit and continues processing. Before detection, the fault can produce an error with constant rate ρ. Should the error be detected with probability q, the presence of the fault is recognized and recovery can still occur. This partial CARE model assumes that once detected, the fault is covered all the time. This assumption is demonstrated by the parameter P_A. Should the error not be detected, it propagates through the system model and causes system failure. This system failure is a conservative modeling assumption and is made to simplify the model because these failure conditions are typically improbable. The Permanent Fault state represents the C exit from the FEHM leading to a degraded state, and the FAIL state corresponds to the S exit leading to the FSPF state.

For this example the FEHM probabilities (ref. 7), when replaced by a branch point, are

$$c = \frac{\delta}{\delta + \rho} + \frac{\rho}{\delta + \rho} q$$

(probability that we take the path from the Active Fault state directly to the Detected state multiplied by P_A) + (probability of taking the path to Active Error multiplied by the probability of going from Active Error to Detected multiplied by P_A) and

$$s = \frac{\rho}{\delta + \rho}(1 - q)$$

(probability of taking path from Active Fault state to Active Error state multiplied by the probability of going from Active Error to FAIL state). However, the HARP user does not have to calculate these coverage factors; they are automatically computed by HARP based on user-specified parameters.

Using *tdrive* to input this example in HARP, we specify the fault tree or Markov chain as in previous examples. However, we now enter the FEHM model in the dictionary. We present this example as a fault tree; however it can just as simply be entered as a Markov chain. (See section 3.3.) As previously noted, the *.DIC, *.TXT, *.INT, *.MAT, *.SYM, and *.RS1 files are given in appendix A.

5.1. *tdrive* Dialog for Input of CARE III Permanent Single Fault Model

```
$  tdrive
                        HARP---Version 7.0, February 1993
                    NASA Langley Research Center/Duke University
                                Program Tdrive
            Defaults are Invoked by "CR", Inputs are Case Insensitive
                        Question?  ( "?" or "help" )
                    FAULT TREE (F) or Markov Chain (M)?
> f
Modelname?
> 3pcare1
 NAME for component ID    1.  Enter  "/d" or "done" if finished.
> processor
Symbolic failure rate?
> lambda
Component FEHM?
> care
FEHM filename?
> care1.fhm
File CARE1.FHM     does not currently exist
 Create now? (y/n) ?  y
****************************************
* CARE III SINGLE FAULT MODEL (MARKOV) *
****************************************
All time parameters should be given in terms of HOURS. If you need help
or additional information, type "HELP" when prompted.
Permanent fault probability?
```

```
> 1
Enter the permanent fault model parameters (alpha and beta are both zero,
 as is PB):
Delta? (rate - events/hour)
> 360
Epsilon? (rate - events/hour)
> 3600
Rho? (rate - events/hour)
> 180
 PA? (0 <= PA <= 1):
> 1.0
 Q? (0 <= Q <= 1):
> .999
FEHM information for this component is stored in file CARE1.FHM
Continue => Y   Reenter => N
> y
 NAME for component ID    2.  Enter  "/d" or "done" if finished.
> done
Define interfering component types for near-coincident faults? (Y/N)?
> no
                        [Not asked in PC 16-bit HARP version.
                        It's significance is explained in the
                        section on near-coincident faults.]
                            Fault Tree Description.
Enter "/d" or done" for gate/box entry, ? for dictionary,
 or "/X" to correct input error.
Basic event node  1:
Component ID?
> 1
Replication factor?
> 3
Summary:  Basic event node   1:   3 of component  1
Continue => Y  Reenter => N, (Default = Y)
> y
Enter "/d" or done" for gate/box entry, ? for dictionary,
```

```
 or "/X" to correct input error.
Basic event node  2:
Component ID?
> done
Enter "/X" to correct input error, ? for help.
   Node   2: Gate or Box or Fbox (Enter "FBOX" as last node)
Enter gate type:
> and
Enter number of incoming arcs:
> 1
 Enter ID number of source node for arc    1:
> 1
 SUMMARY:  Node   2: TYPE AND          ,   1 INPUTS:   1
Continue => Y  Reenter => N, (Default = Y)
> y
Enter "/X" to correct input error, ? for help.
   Node   3: Gate or Box or Fbox (Enter "FBOX" as last node)
Enter gate type:
> fbox
 Enter ID number of source node for arc    1:
> 2
Summary: FBOX node  3:  INPUT:    2
Continue => Y  Reenter => N, (Default = Y)
> y
TRUNCATE the model after how many failures?
> 0
Default selected: no truncation.
Include state tuples as comments in .INT file?
> n
Default selected: No state tuple notation.
FT2MC: Converting fault tree to Markov chain . . .
FT2MC: Successful completion
            4 internal Markov chain states generated
            3 unique nonfailure states
            1 failure states generated for HARP engine
```

```
Model information in file:      3PCARE1.INT
Dictionary information in file: 3PCARE1.DIC
```

5.2. *fiface* Dialog for CARE III Permanent Single Fault Model

```
$  fiface
                        HARP---Version 7.0, February 1993
                                Program FIFACE
Modelname?
> 3pcare1
Matrix and symbol table information in: 3PCARE1.MAT
Which near-coincident fault rate files are to be created?
Enter:
N for NONE (ignore near-coincident faults)
A for ALL (all near-coincident faults are fatal)
S for SAMe (only faults of same type interfere)
U for USeR defined interfering component types
You can type combinations like AU, ASU, SA etc.
Combinations of "N" with A, U or S are not allowed.
                    [Not asked in PC 16-bit HARP version.
                    It's significance is explained in the
                    section on near-coincident faults.]
> no
Not creating any near-coincident fault rate files.
```

5.3. *harpeng* Dialog for Solution of CARE III Permanent Single Fault Model

```
$ harpeng
                ----------------- HARP -----------------------
                - The Hybrid Automated Reliability Predictor -
                ------------ Release Version 7.0 -------------
                -------------- February 1993 -----------------
 Use an echo file from a previous run as the input file?  y/n ?
> no
 Modelname ?
> 3pcare1
 Output files:
```

```
     3PCARE1.RS1     --  Reliability and state probabilities
     3PCARE1.PT1     --  Graphics information
                              ----- WORKING -----
     3PCARE1.INP     --  Input file or echo of input
 Declare meaning for symbol  LAMBDA        ( "?" or "help" )
> 1
 For constant failure rate: LAMBDA
 Nominal value?
> 0.001
 (+/-) Variation? (Must be less than nominal.          "?" will allow reentry. )
> 0
 Redefine symbol(s) meanings or their values, or correct an error (y/n)?
> no
 Mission time? (Hours):
> 10
 Mission time reporting interval? (Hours):
> 10
Calculating State Probabilities...
 There were        0 Warnings from the GERK ODE solver.
 Please select:
 1: Scroll through the result file
 2: Solve same model with new mission time or near-coincident fault rates, etc.
 3: Redefine symbol(s) meaning(s) and re-run model
 4: Exit the program.
> 4
```

Chapter 6

Modeling Transient Faults and Transient Recovery

Until now, we have assumed that the only faults to be modeled are permanent. However, certain faults can be temporary in nature and not cause permanent physical damage but still result in software errors. These faults are called *transient faults* and the effect on the FORM of such faults is represented in figure 14 as the transient restoration transition. Once a fault is diagnosed as transient and recovery from such a fault is successful, the system returns to an operational mode without reconfiguring the system, that is, a hardware module is not removed from the system. Transient faults can be modeled by using the Direct ARIES, CARE III, and ESPN FEHM's. These models allow the user to model specific system behaviors resulting from the occurrence of transient faults. The particular choice of FEHM depends on the system application and its susceptibility to transients. The other FEHM's, the probability and moments, probability and distributions, and probability and empirical data, can also be used to account for transient faults, but no FEHM modeling detail is allowed.

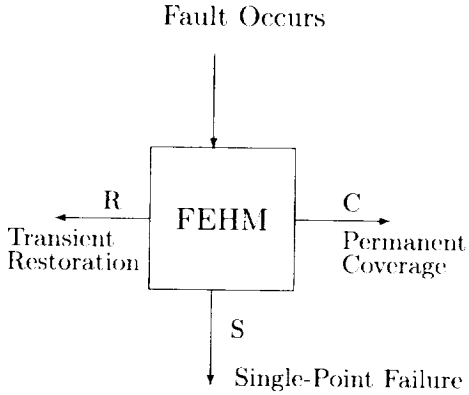


Figure 14. Partial structure of HARP FEHM.

The incorporation of transient faults in our three-processor example is shown in figure 15. While this figure appears similar to figure 9, the boxes now show a transition back to the state from which the box was entered, corresponding to transient restoration.

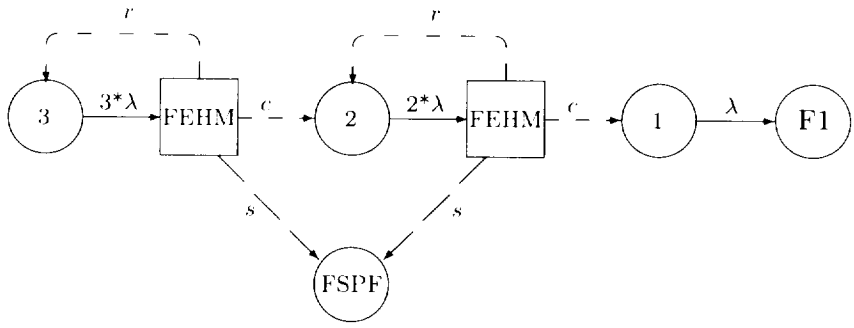


Figure 15. Three-processor system with FEHM's showing the C, S, and R exits.

6.1. Direct Coverage Values Model

If the user chooses, the coverage values can be input directly either in the *fiface* program or the *harpeng* program. In this case the FEHM type is VALUES. When prompted for the FEHM type in program *tdrive*, the user should respond with the keyword VALUES. No input file is used; instead, in *fiface* the user is given the option of entering the specific values for C and R. If they are not entered in *fiface*, they are requested in *harpeng*. The value for S is calculated as $(1 - C - R)$.

6.2. ARIES Transient Recovery Model

To demonstrate the modeling of transient faults, assume the boxes in figure 15 now represent the ARIES model (ref. 9). (See fig. 16.) ARIES is a phased recovery model that allows the user to specify how many phases comprise the recovery procedure.

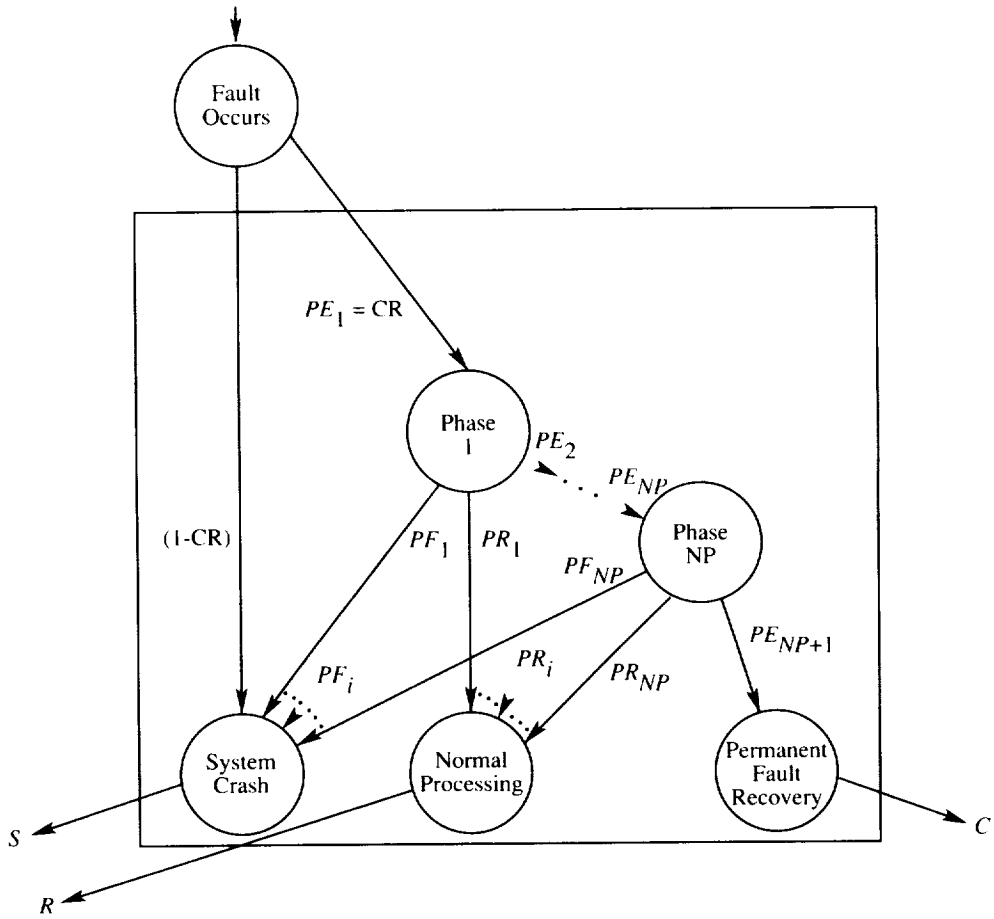


Figure 16. ARIES transient fault recovery model.

In each phase of the recovery, the duration of which is constant, the system attempts recovery. If successful, the system returns to the Normal Processing state without discarding any components. If the recovery in a particular phase is unsuccessful, the next phase attempts to locate and recover from the fault. If all phases are ineffective, the fault is assumed to be permanent. We discard the faulty component and continue running with one fewer component (provided that we still have enough to leave the system operational). If the fault is a critical

one from which the system as a whole cannot recover, the System Crash state is entered by the transition arc labeled (with probability) $1 - CR$. This state represents the S exit from the FEHM and the "Permanent Fault Recovery" state represents the C exit. The transient faults, leaving the FEHM via exit R, are realized by state Normal Processing.

The exit probability calculations[8] for the ARIES model are as follows:

$$c = (PE_{NP+1})(cov)$$

$$s = (1 - CR) + (PE_{NP+1})(1 - cov) + \sum_{i=1}^{NP} PF_i$$

$$r = \sum_{i=1}^{NP} PR_i$$

where

$$PF_i = PE_i\left(1 - \exp^{-\rho Ti}\right)$$

These calculations are performed by the HARP program and not by the user. The user-input data for the ARIES model are delineated in the following example as annotations enclosed in square brackets. For this example, we provide only the coverage model input. Like the previous example, the model is input in program *tdrive*. Because the output files from *tdrive* and *fiface* are similar to previous runs, the *harpeng* output file 3PARIES1.RS1 and the FEHM file ARIES.FHM are listed in appendix A.

6.3. *tdrive* Dialog for Input of ARIES Model

```
**************************************
* ARIES TRANSIENT FAULT RECOVERY MODEL *
**************************************
 Enter number of recovery phases (int, max 10):  [NP]
> 3
 Transient fault probability?
> .9
 Transient fault mean duration?  (in seconds):
> .005
 Catastrophic fault probability, given that a fault occurs?  [1-CR]
> .001
Duration of each recovery phase? (seconds): [Ti, deterministic time,
                                             conservative assumption]
 Phase  1:
> .8
```

[8] Parameter *cov* is an enhancement to the original ARIES FEHM model in recognition that after the system determines the fault is permanent, some recovery action is necessary. The success of that action is specified by *cov*, a probability.

```
 Phase  2:
> .2
 Phase  3:
> .1
Recovery effectiveness probability of each phase? [PR(i)]
 Phase 1:
> .8
 Phase 2:
> .7
 Phase 3:
> .5
Failure rate of the recovery system hardware (in seconds):  [ρ]
0.0
 Coverage of permanent fault recovery procedure: (probability)  [cov]
> .85
FEHM information for this component is stored in file ARIES.FHM
```

6.4. CARE III Transient Single Fault Model

The CARE III single fault model can be expanded to model transient and intermittent faults. As figure 17 shows, permanent faults are still modeled in the same manner as previously described, with $\alpha = \beta = P_B = 0$. (The user is never permitted to enter these default parameters.) For the transient model, the fault can now be either active or benign. Once the fault enters the Benign Fault state, it is assumed to have disappeared before the system experienced any adverse effects ($\beta = 0$). The disappearance of the transient signifies that the FEHM exit R is being taken. Again, in the Active Error state, the transient can go benign. If the error is detected (with probability q), the faulty element is removed from service with probability P_B.

With the complementary probability, the fault is assumed to be transient and the element is returned to service without reconfiguring the system. If the error is detected from the Active Error state, the faulty element is removed from service with probability P_A. With the complementary probability, we remain in the FEHM because the fault is still present. Note, the two Detected states and the Benign state (for the transient model) are instantaneous states, as denoted by the dotted transitions leaving them. By setting $\alpha > 0$ and $\beta > 0$, we can also model intermittent faults. (See the following section.)

The next section lists the dialog for the input of the coverage model in program *tdrive*: The FEHM file 3PCARE2.FHM and the *harpeng* output file 3PCARE2.RS1 are listed in appendix A. Using the CARE FEHM example to model transient faults, we let the parameter α be large in relation to the values of ρ and δ. Once again, the calculation of the exit probabilities C, R, and S is performed automatically by HARP, and each occurrence of the CARE III FEHM is replaced by a three-way branch point.

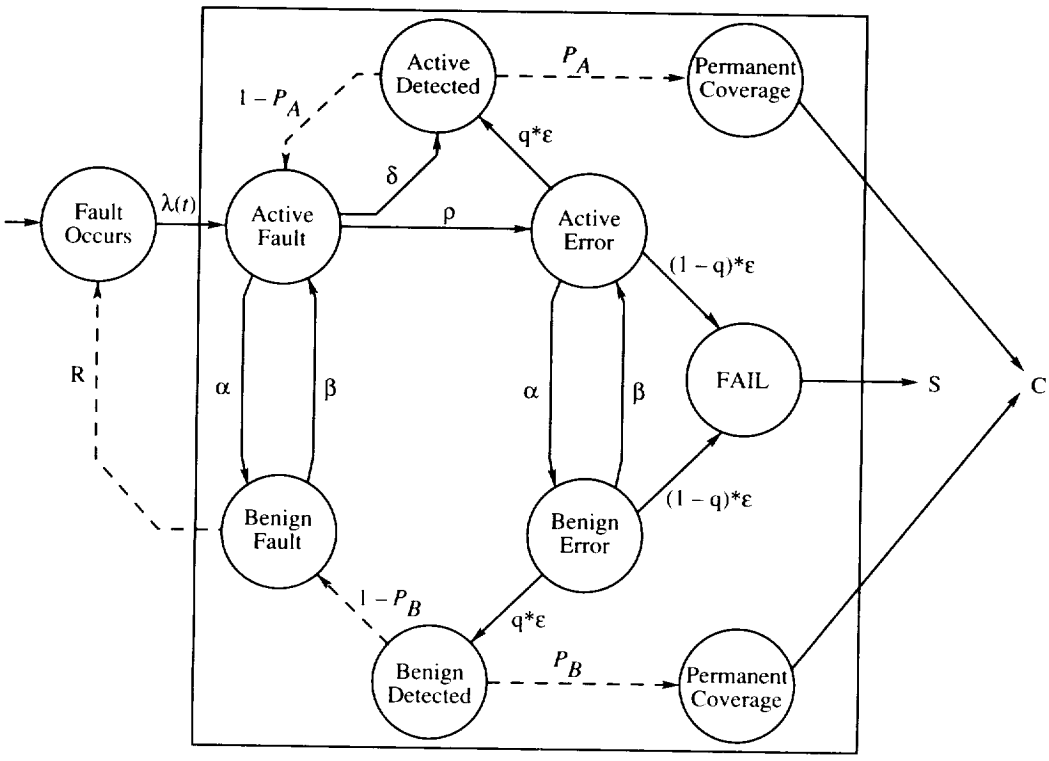


Figure 17. Complete CARE III single fault model.

6.5. *tdrive* Dialog for Input of CARE III Transient Single Fault Model

```
****************************************
* CARE III SINGLE FAULT MODEL (MARKOV) *
****************************************
All time parameters should be given in terms of HOURS. If you need help
or additional information, type "HELP" when prompted.
Permanent fault probability?
> 0
Intermittent fault probability?
> 0
Transient fault probability is:      1.0000000000000
Enter the transient fault model parameters
 (alpha is positive but beta is zero):
Alpha? (rate - events/hour)
> 36000
Delta? (rate - events/hour)
```

```
> 360
Epsilon? (rate - events/hour)
> 3600
Rho? (rate - events/hour)
> 180
 PA? (0 <= PA <= 1):
> .5
 PB? (0 <= PB <= 1):
> .5
 Q? (0 <= Q <= 1):
> .9
FEHM information for this component is stored in file CARE2.FHM
```

Chapter 7

Intermittent Faults in Coverage Model

7.1. Overview

A third class of faults known as *intermittent faults* can be modeled. These faults are particularly insidious as they are always present but not always active. In the active state, the intermittent fault causes the system to operate incorrectly; however, in the benign state, the intermittent fault does not affect the operation of the system. The fault can switch between the active and benign states at any time ($\alpha > 0$ and $\beta > 0$). The FEHM model, like in the transient case, has the C, S, and R exits. (See fig. 14.) The C exit is used when the intermittent is treated as a permanent fault, the S exit is used when the fault has produced an error from which the system cannot recover, and the R exit is used when the intermittent is treated as a transient. (That is, the time between activations of the intermittent fault can be long and results in the incorrect assumption that the fault is a transient.)

The CARE III single fault model again provides us a good example for which we provide the coverage model input. In previous examples demonstrating the CARE III FEHM model, we stated that the particular fault type that we are modeling is going to occur 100 percent of the time; that is, the model is the permanent fault. This model is selected when answering the following questions:

```
"A Fault is Permanent with what probability?   "

"A Fault is Intermittent with what probability?   "
```

For the permanent model, we responded 1.0 to the first question, and for the transient model, we responded 0.0 to both questions (thus making the transient model a default of 1.0). Rather than determining that only one type of fault is likely, perhaps we have studied our system and found that all three fault types are possible. We can reflect this in our model during the input. As in the previous two examples, the FEHM file 3PCARE3.FHM and the *harpcng* output file 3PCARE3.RS1 are given in appendix A.

7.2. *tdrive* Dialog for CARE III Intermittent Single Fault Model

```
****************************************
* CARE III SINGLE FAULT MODEL (MARKOV) *
****************************************
All time parameters should be given in terms of HOURS. If you need help
or additional information, type "HELP" when prompted.
Permanent fault probability?
> .2
Intermittent fault probability?
> .2
Transient fault probability is:     0.60000000000000
```

```
Enter the permanent fault model parameters (alpha and beta are both zero,
 as is PB):
Delta? (rate - events/hour)
> 300
Epsilon? (rate - events/hour)
> 3600
Rho? (rate - events/hour)
> 240
 PA? (0 <= PA <= 1):
> 1
 Q? (0 <= Q <= 1):
> .999
Enter the intermittent fault model parameters:
Alpha? (rate - events/hour)
> 2100
Beta? (rate - events/hour)
> 3000
Delta? (rate - events/hour)
> 360
Epsilon? (rate - events/hour)
> 3600
Rho? (rate - events/hour)
> 180
 PA? (0 <= PA <= 1):
> .9
 PB? (0 <= PB <= 1):
> .1
 Q? (0 <= Q <= 1):
> .999
Enter the transient fault model parameters
 (alpha is positive but beta is zero):
Alpha? (rate - events/hour)
> 36000
Delta? (rate - events/hour)
> 180
```

```
Epsilon? (rate - events/hour)
> 3600
Rho? (rate - events/hour)
> 180
 PA? (0 <= PA <= 1):
> .5
 PB? (0 <= PB <= 1):
> .5
 Q? (0 <= Q <= 1):
> .999
FEHM information for this component is stored in file CARE3.FHM
```

Chapter 8

ESPN FEHM

One additional coverage model is available to the user, an Extended Stochastic Petri Net (ESPN). (See refs. 10 to 13.) As shown in figure 18, this FEHM models three aspects of a fault recovery process: physical fault behavior, transient recovery, and permanent recovery.

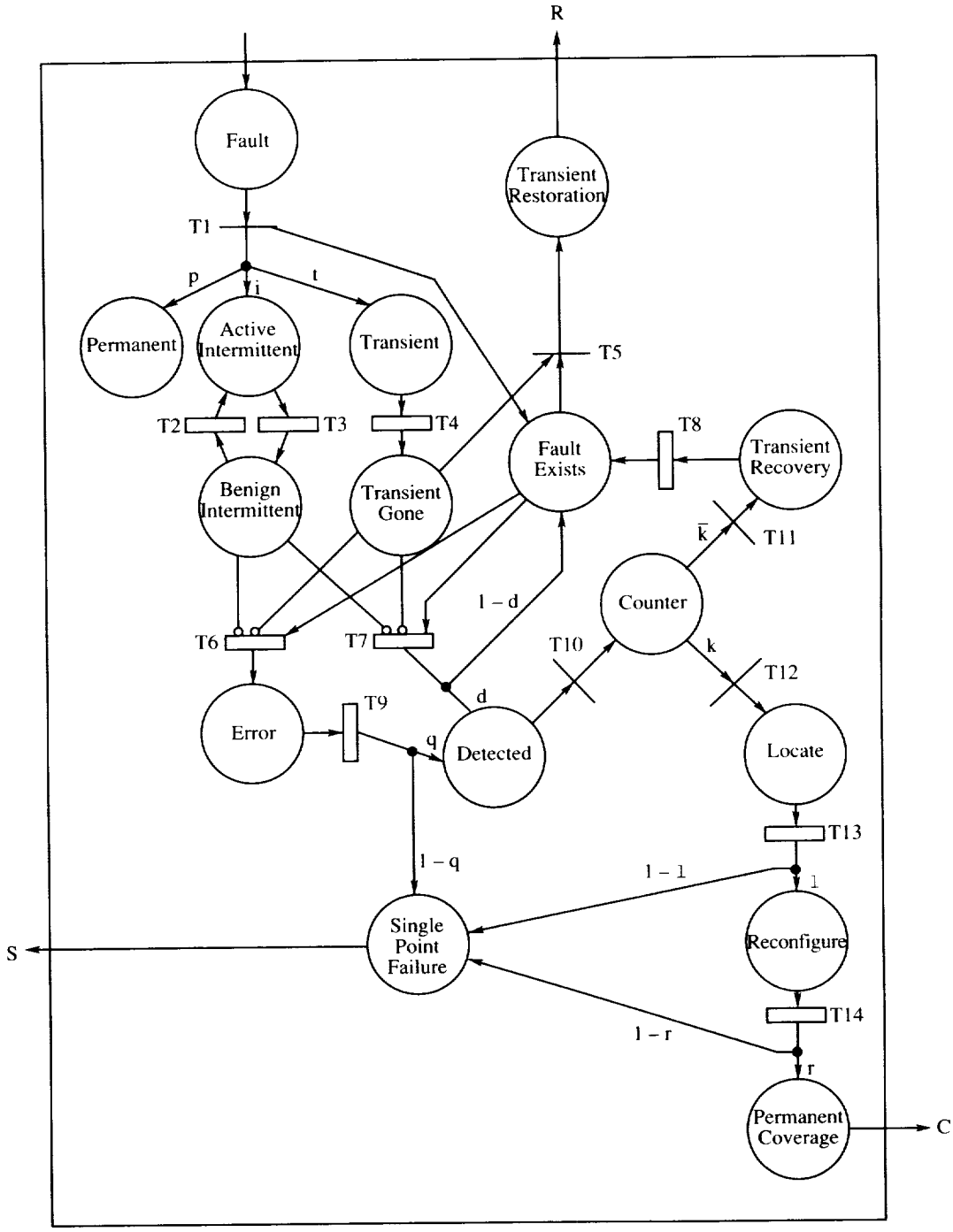


Figure 18. HARP ESPN single fault model.

The fault behavior model captures the physical status of the fault, such as whether the fault is active or benign (if permanent or intermittent), or whether the fault still exists (if transient). Once the fault is detected, it is temporarily assumed to be transient, and an appropriate recovery procedure can commence. The transient recovery procedure can be attempted more than once. If the detection/recovery cycle is repeated too many times, a permanent recovery procedure (reconfiguration) is invoked. If the reconfiguration is successful, the system is again operating correctly, although in a somewhat degraded state.

8.1. ESPN Specification

The inherent concurrency between the actual fault behavior and the system's fault/error handling behavior can be captured effectively in terms of an ESPN (ref. 14). Recall the composition of a Petri net (PN) bipartite graph: (ref. 15) a set of places P (drawn as circles), a set of transitions T (drawn as bars), and a set of directed arcs A, which connect transitions to places or places to transitions. Places can contain *tokens*[9] (drawn as dots). The state of a PN, called the PN marking, is defined by the number of tokens contained in each place.

A place is an input to a transition when an arc exists from the place to the transition, and a place is an output from a transition when an arc exists from the transition to the place. A transition is *enabled* when each of its input places contains at least one token. Enabled transitions can fire, by removing one token from each input place and placing one token in each output place. Thus, the firing of a transition causes a change of state (produces a different marking) for the PN.

A Stochastic Petri Net (ref. 16) is obtained by associating with each transition a so-called firing time. Once a transition is enabled, an exponentially distributed amount of time elapses. If the transition is still enabled, it then fires. A Generalized Stochastic Petri Net (ref. 17) allows immediate (zero firing time) as well as timed transitions. Immediate transitions are drawn as thin bars, timed transitions as thick bars.

An ESPN allows firing times to belong to an arbitrary distribution. Some other extensions to Petri nets are considered here. An *inhibitor arc* from a place to a transition has a small circle rather than an arrowhead at the transition. The firing rule is changed as follows. A transition is enabled when tokens are present in all of its (normal) input places and no tokens are present in the inhibited input places. When the transition fires, the tokens are removed from the normal input places and deposited in the output places as usual, but the number of tokens in the inhibited input place remains zero.

A *probabilistic arc* from a transition to a set of output places deposits a token in one (and only one) of the places in the set. The choice of which place receives the token is determined by the probability labels on each branch of the arc.

A *counter arc* from a place to a transition is labeled with an integer value k. This the integer value changes the firing rule such that a transition is enabled when tokens are present in all of its (normal) input places and at least k tokens are present in the counter input place. When the transition fires, one token is removed from each normal input place, while k tokens are removed from the counter input place. Associated with a particular counter arc can be a *counter alternate arc*, which enables an alternate transition when the count is between 1 and $k-1$, inclusive. The alternate transition can fire each time a token is deposited in the counter input place until k tokens are present. The count remains unchanged by the firing of the alternate transition because it removes no token from the counter input place. A counter alternate arc is labeled with a $\bar{k}$. Neither the counter arc nor the counter alternate arc are true extensions to Petri

[9] A token is a marker that designates flow of model processes.

nets, as both can be realized by a cascade of normal places and transitions. Rather, the arcs are useful shorthand notations for such a cascade.

8.2. ESPN Model in HARP

When a fault occurs in the system, a token is deposited in the place labeled *Fault*. This token enables the transition $T1$. The transition fires immediately, thus removing a token from the input place. Depending upon whether the fault is permanent, intermittent, or transient, a token is then deposited in place Permanent, Active Intermittent, or Transient, with probability p, i, or t, respectively. (These probabilities are user-input values.) Simultaneously, a token is deposited in place Fault Exists, which represents the presence of an as yet undetected fault. If the fault is permanent, the token remains in the Permanent place until the model is exited. If the fault is intermittent, the token that was deposited in Active Intermittent circulates between places Active Intermittent and Benign Intermittent, thus representing the oscillation of the fault between the active and benign states. If the fault is transient, eventually the token that was deposited in place Transient is passed to place Transient Gone, which represents the disappearance of the fault. Note that if a token exists in both places Transient Gone and Fault Exists, transition $T5$ can fire. This condition represents a transient fault that disappears before its presence is felt.

While the fault is active and still exists (i.e., a token exists in place Fault Exists and no token in either places Benign Intermittent or Transient Gone), two things can happen: an error can be produced or the fault can be detected directly. These two events are represented by transitions $T6$ and $T7$, respectively. If the self-test procedure is run while the fault is active, then the fault is detected with probability d. Once an error is produced, it is detected with probability q, or it propagates through the system and causes a system failure.

Once the fault is detected, a token is deposited in place Counter, which serves as a counter for the number of times transient recovery has been attempted. As long as fewer than k tokens are in place Counter, transient recovery can begin. When recovery is completed, the fault can still exist, and the detection/recovery cycle can repeat. If recovery is completed and the transient fault is gone, $T5$ fires, and the system is once again functioning correctly. If the recovery has completed and the intermittent fault has become benign, transitions $T6$ and $T7$ wait for the fault to become active again before they are enabled.

If the fault is detected too often (more than k times), the fault is then assumed to be permanent in nature, and no automatic recovery process begins. This condition is modeled by the accumulation of k tokens in place Counter. Once k tokens are present, transition $T11$ is disabled (transient recovery procedures are inhibited) and transition $T12$ is enabled (permanent recovery procedures begin). Once the fault is determined to be permanent, a diagnostic procedure is invoked to isolate the faulty unit; this condition is represented by a token in place Locate. The diagnostic procedure is successful with probability l. If the faulted unit is isolated, the system attempts automatic reconfiguration, which is represented by place Reconfigure. Reconfiguration is successful with probability r and the token is passed to place Permanent Coverage, which represents the system again operating correctly, although performance can be somewhat degraded.

The user input to this submodel are the distributions of times for each transition, and the probabilities of correct error detection q, fault detection d, fault location l, and reconfiguration r. (Note that the distributions need not be exponential.) The user must also provide the number of attempts at transient recovery $k-1$, the percentage of faults that are permanent p, the percentage of faults that are transient t, and, since this model is simulated for solution the confidence level and percent error desired. The distributions available are constant, k-stage

Erlang, exponential, gamma, log-normal, normal, Rayleigh, uniform, and Weibull. For more information on any of these distributions, see reference 18.

This FEHM is the only model that is simulated for solution. During the simulation, a statistical analysis of the simulation data is performed. The confidence intervals about the exit probabilities are generated for the R and C exits and are compared with the allowable error. The S exit data is determined to be $S = 1 - R - C$. If the confidence interval is too wide, the number of trials is increased (by a factor of 2). When the simulation has reached the desired accuracy, the results are appended to the parameter file. The ESPN simulator uses a random number generator whose seed is linked to the host system clock. Thus, model state probabilities change with each new execution of HARP, even when the same input data are used. The user cannot replicate the results for the following example, which is listed in the appendix, unless the random number seed is set. (See vol. 1 of this TP.)

For this model, the coverage factor R is the probability of a token reaching the place labeled "Transient Restoration"; C is the probability of a token reaching the place labeled "Permanent Coverage"; and S is the probability of a token reaching the place labeled "Single Point Failure." The fourth factor, N is derived from the relative passage time to the three exits, as has been described previously.

To demonstrate the use of the ESPN model, the three-processor, two-bus system is used. The FORM input is left to the reader (either as a fault tree or a Markov chain) and the output files are listed in the appendix. For this example the *harpeng* program is run four times utilizing the four different near-coincident fault type options. The results for the four runs are recorded in the files with extensions .RS1, .RS2, .RS3, and .RS4. Like our first examples, the complete program runs are listed along with a sample input ESPN model. The ESPN parameter file is printed twice in appendix A both before and after the solution program is run. The simulation results obtained during the execution of *harpeng* are printed directly in the parameter file. In this way, unless the input parameters change, the simulation is not run again.

8.2.1. *tdrive* Dialog for ESPN Model

```
$  tdrive

                    HARP---Version 7.0, February 1993

                NASA Langley Research Center/Duke University

                            Program Tdrive

        Defaults are Invoked by "CR", Inputs are Case Insensitive

                            Question?  ( "?" or "help" )

                    FAULT TREE (F) or Markov Chain (M)?

>  f

Modelname?

>  3p2b

 NAME for component ID    1.  Enter  "/d" or "done" if finished.

> processor

Symbolic failure rate?

> lambda

Component FEHM?
```

```
> espn
FEHM filename?
> espn.fhm
File ESPN.FHM      does not currently exist
 Create now? (y/n) ?  y
************************************
*     HARP ESPN COVERAGE MODEL     *
************************************
 All times are in units of SECONDS
Transition numbers refer to ESPN figure in manual
Active to benign transition distribution? (T3)
    Distribution type:
> help
 Valid dists are:
      uniform
      exponential
      Weibull
      normal
      Rayleigh
      log Normal
      Erlang (k-stage Erlang)
      constant value
 please try again
    Distribution type:
> unif
       Lower limit (seconds):
> 0
       Upper limit (seconds):
> 1
Transient fault lifetime distribution? (T4)
    Distribution type:
> exp
       lambda (rate parameter, events/second):
> 100
Benign to active transition distribution? (T2)
```

```
    Distribution type:
> unif
        Lower limit (seconds):
> 0
        Upper limit (seconds):
> .5
Detect transition distribution (self-test)? (T7)
    Distribution type:
> unif
        Lower limit (seconds):
> 0
        Upper limit (seconds):
> .4
Fraction of faults detected (d)?
> .9
Production of errors distribution? (T6)
    Distribution type:
> weibull
        Scale parameter (rate, events/second):
> 10
Shape parameter? (alpha)
> 2.5
Error propagation or detection distribution? (T9)
    Distribution type:
> weib
        Scale parameter (rate, events/second):
> 50
Shape parameter? (alpha)
> .25
Fraction of errors detected? (q)
> .9
Transient recovery attempts? (k-1)
> 5
Transient recovery distribution? (T8)
    Distribution type:
```

```
> erlang
      Rate parameter (events/second):
> 100
      Number of stages (positive integer):
> 2
Fraction of isolated detected faults? (l)
> .9
Isolation time distribution? (T13)
    Distribution type:
> normal
      Mean (seconds):
> 4
      Standard deviation (seconds):
> 1
Fraction of successful reconfigurations? (r)
> .9
Reconfiguration time distribution? (T14)
    Distribution type:
> normal
      Mean (seconds):
> 1
      Standard deviation (seconds):
> .5
Fraction of transient faults? (t)
> .5
Fraction of permanent faults? (p)
> .4
Confidence level? (choose from 60,65,70,75,80,85,90,95,98,
 -- suggested value is 95)
> 90
Percent error tolerated in the exit probabilities? (integer value --
 suggest value between 2 and 5)
> 10
FEHM information for this component is stored in file ESPN.FHM
Continue => Y   Reenter => N
```

```
> y
 NAME for component ID    2.  Enter  "/d" or "done" if finished.
> bus
Symbolic failure rate?
> mu
Component FEHM?
> values
Continue => Y   Reenter => N
> y
 NAME for component ID    3.  Enter  "/d" or "done" if finished.
> done
Define interfering component types for near-coincident faults? (Y/N)?
> y
   1 PROCESSOR     LAMBDA        ESPN.FHM
   2 BUS           MU            VALUES
When prompted for each component,enter the number of each dictionary ID
that is an interfering component type.
Separate entries by commas, i.e., 1,2.
   Type "ALL" to specify all components.
   Type "NONE" to specify no components.
   Type "?" or"HELP" to see the dictionary again.
 What components will cause the PROCESSOR     to fail
> 2
                          Fault Tree Description.
Enter "/d" or done" for gate/box entry, ? for dictionary,
 or "/X" to correct input error.
Basic event node  1:
Component ID?
> 1
Replication factor?
> 3
Summary:  Basic event node   1:   3 of component  1
Continue => Y  Reenter => N, (Default = Y)
> y
Enter "/d" or done" for gate/box entry, ? for dictionary,
```

```
 or "/X" to correct input error.
Basic event node  2:
Component ID?
> 2
Replication factor?
> 2
Summary:  Basic event node   2:   2 of component  2
Continue => Y  Reenter => N, (Default = Y)
> y
Enter "/d" or done" for gate/box entry, ? for dictionary,
 or "/X" to correct input error.
Basic event node  3:
Component ID?
> done
Enter "/X" to correct input error, ? for help.
   Node   3: Gate or Box or Fbox (Enter "FBOX" as last node)
Enter gate type:
> and
Enter number of incoming arcs:                * Since replication = 3 *
                                              * was specified, only 1 *
> 1                                           * arc is given here.    *
 Enter ID number of source node for arc    1:
> 1
 SUMMARY:  Node   3: TYPE AND          ,   1 INPUTS:   1
Continue => Y  Reenter => N, (Default = Y)
> y
Enter "/X" to correct input error, ? for help.
   Node   4: Gate or Box or Fbox (Enter "FBOX" as last node)
Enter gate type:
> and
Enter number of incoming arcs:                * See note directly above *
> 1
 Enter ID number of source node for arc    1:
> 2
 SUMMARY:  Node   4: TYPE AND          ,   1 INPUTS:   2
```

```
 Continue => Y  Reenter => N, (Default = Y)
 > y
 Enter "/X" to correct input error, ? for help.
    Node   5: Gate or Box or Fbox (Enter "FBOX" as last node)
 Enter gate type:
 > or
 Enter number of incoming arcs:
 > 2
  Enter ID number of source node for arc     1:
 > 3
  Enter ID number of source node for arc     2:
 > 4
  SUMMARY:  Node   5: TYPE OR           ,   2 INPUTS:   3  4
 Continue => Y  Reenter => N, (Default = Y)
 > y
 Enter "/X" to correct input error, ? for help.
    Node   6: Gate or Box or Fbox (Enter "FBOX" as last node)
 Enter gate type:
 > fbox
  Enter ID number of source node for arc     1:
 > 5
 Summary: FBOX node  6:  INPUT:    5
 Continue => Y  Reenter => N, (Default = Y)
 > y
 TRUNCATE the model after how many failures?
 > 0
 Default selected: no truncation.
 Include state tuples as comments in .INT file?
 > n
 Default selected: No state tuple notation.
 FT2MC: Converting fault tree to Markov chain . . .
 FT2MC: Successful completion
           11 internal Markov chain states generated
            6 unique nonfailure states
            2 failure states generated for HARP engine
 Model information in file:      3P2B.INT
 Dictionary information in file: 3P2B.DIC
```

8.2.2. *fiface* Dialog for ESPN Model

```
$  fiface
                        HARP - Version 7.0, February 1993
                                Program FIFACE
Modelname?
> 3p2b
Matrix and symbol table information in: 3P2B.MAT
Which near-coincident fault rate files are to be created?
Enter:
N for NONE (ignore near-coincident faults)
A for ALL (all near-coincident faults are fatal)
S for SAMe (only faults of same type interfere)
U for USeR defined interfering component types
You can type combinations like AU, ASU, SA etc.     *If more than one multi- *
Combinations of "N" with A, U or S are not allowed.*fault model is required *
                                                    *when multiple harpeng   *
                                                    *executions are made,    *
                                                    *specify them here. fiface*
> asu                                               *will create .ALL,.SAM,  *
                                                    *or .USR files for harpeng*
Enter probabilities now for component with failure rate MU? (y/n) ?
> y
The upper bounds of C and R and lower bounds of S and N should add to one.
The lower bounds of C and R and upper bounds of S and N should add to one.
Also the nominal values of C, N, R, S should add to 1.
Probability of C2 ?
> .5
Variation?
> 0
Probability of R2 ?
> .3
Variation?
> 0
Probability of S2 ?
> .2
Variation?
> 0
 Probability of N calculated to be:  0.000000
 Variation of N calculated to be:    0.000000
```

8.2.3. *harpeng* Dialog for ESPN Model

```
$ harpeng

                ---------- HARP ----------
                - The Hybrid Automated Reliability Predictor -
                ---------- Release Version 7.0 ----------
                ---------- February 1993 ----------
 Use an echo file from a previous run as the input file?  y/n ?
> no
 Modelname ?
> 3p2b
 Output files:
     3P2B.RS1        --  Reliability and state probabilities
     3P2B.PT1        --  Graphics information
                             ----- WORKING -----
     3P2B.INP        --  Input file or echo of input
 Choose the near-coincident fault rate to be used
 for the coverage factor calculations.
 1: NONE (ignore near-coincident faults).
 2: ALL-inclusive (all near-coincident faults are fatal)
 3: SAMe-component (only faults of same type interfere).
 4: Interfering component types (USeR-defined types).
> 1
 Declare meaning for symbol  LAMBDA        ( "?" or "help" )
> 1
 For constant failure rate: LAMBDA
 Nominal value?
> .5e-2
 (+/-) Variation? (Must be less than nominal.          "?" will allow reentry. )
> 0
 Declare meaning for symbol  MU            ( "?" or "help" )
> 1
 For constant failure rate: MU
 Nominal value?
> .5e-1
 (+/-) Variation? (Must be less than nominal.          "?" will allow reentry. )
```

```
> 0
 Redefine symbol(s) meanings or their values, or correct an error (y/n)?
> n
 Mission time? (Hours):
> 10
 Mission time reporting interval? (Hours):
> 10
 Compute Parametric Bounds using SIMPLE Model? (y/n) ?  n
Simulating ESPN Fault/Error Handling Model ...
Calculating State Probabilities...
        1 Reports from the GERK ODE solver.
 Please select:
 1: Scroll through the result file?
 2: Solve same model with new mission time or near-coincident fault rates, etc.?
 3: Redefine symbol(s) meaning(s) and re-run model?
 4: Exit the program?
> 2
 Choose the near-coincident fault rate to be used
 for the coverage factor calculations.
 1: NONE (ignore near-coincident faults).
 2: ALL-inclusive (all near-coincident faults are fatal)
 3: SAMe-component (only faults of same type interfere).
 4: Interfering component types (USeR-defined types).
> 2
 Redefine symbol(s) meanings or their values, or correct an error (y/n)?
> n
 Mission time? (Hours):
> 10
 Mission time reporting interval? (Hours):
> 10
 Compute Parametric Bounds using SIMPLE Model? (y/n) ?  n
 Calculating State Probabilities...
        1 Reports from the GERK ODE solver.
 Please select:
 1: Scroll through the result file?
```

```
 2: Solve same model with new mission time or near-coincident fault rates, etc.?
 3: Redefine symbol(s) meaning(s) and re-run model?
 4: Exit the program?
> 2
 Choose the near-coincident fault rate to be used
 for the coverage factor calculations.
 1: NONE (ignore near-coincident faults).
 2: ALL-inclusive (all near-coincident faults are fatal)
 3: SAMe-component (only faults of same type interfere).
 4: Interfering component types (USeR-defined types).
> 3
 Redefine symbol(s) meanings or their values, or correct an error (y/n)?
> n
 Mission time? (Hours):
> 10
 Mission time reporting interval? (Hours):
> 10
 Compute Parametric Bounds using SIMPLE Model? (y/n) ?  n
Calculating State Probabilities...
        0 Reports from the GERK ODE solver.
 Please select:
 1: Scroll through the result file?
 2: Solve same model with new mission time or near-coincident fault rates, etc.?
 3: Redefine symbol(s) meaning(s) and re-run model?
 4: Exit the program?
> 2
 Choose the near-coincident fault rate to be used
 for the coverage factor calculations.
 1: NONE (ignore near-coincident faults).
 2: ALL-inclusive (all near-coincident faults are fatal)
 3: SAMe-component (only faults of same type interfere).
 4: Interfering component types (USeR-defined types).
> 4
 Redefine symbol(s) meanings or their values, or correct an error (y/n)?
> n
```

```
 Mission time? (Hours):
> 10
 Mission time reporting interval? (Hours):
> 10
 Compute Parametric Bounds using SIMPLE Model? (y/n) ?  n
Calculating State Probabilities...
        0 Reports from the GERK ODE solver.
 Please select:
 1: Scroll through the result file?
 2: Solve same model with new mission time or near-coincident fault rates, etc.?
 3: Redefine symbol(s) meaning(s) and re-run model?
 4: Exit the program?
> 4
```

Chapter 9

Incorporation of Near-Coincident Faults

9.1. Overview

HARP is designed with the capability to model highly reliable systems. To appropriately do so, the possibility of modeling the effects of near-coincident faults is included. A near-coincident fault is one that occurs before the coverage model has recovered from a single fault. How disastrous the results are depends upon how the user chooses to interpret the effects of the near-coincident fault. Typically, the user models the effect of a near-coincident fault as a system failure. This conservative assumption is often used to eliminate the user burden of acquiring hard-to-get data and to simplify the model. HARP offers a number of multifault models to cover the near-coincident fault effect, that is, system failure.

The FEHM's are specified in the same manner as before, supplying the C, S, and R exit probabilities. Until now, these exit probabilities have been obtained with no time limit on the recovery procedure. However, if a second fault occurs before reaching an exit then we are faced with the problem of two existing faults. Because the second fault can crash the system, we ideally want the FEHM (coverage model) to exit *before* the second, near-coincident fault occurs; however, for highly reliable systems, the probability of a second fault occurring in the recovery interval is often a significant portion of the total system failure probability. The near-coincident fault model allows the user to account for pairs of faults that are likely to cause total system failure.

When these models were being developed over a decade ago, the developers believed that a more complex model allowing more than two near-coincident faults would be of little practical use and would not justify the additional computational burden for the aircraft flight control application. As electronic devices became more reliable during that decade and continue to do so, the developers' assumption proved correct. Most commercial and military aircraft flight control systems and most existing systems in commercial use today can be effectively modeled when the near-coincident fault is a mission critical factor. Systems using computers can have up to four reconfigurable processing units where a majority vote can be effected until two coexisting faults occur. When systems incorporate more than four voting processors and the near-coincident fault is a significant factor, the HARP multifault models produce a conservative approximation that becomes more conservative as the number of processors increases.

During that same decade, electronic microcomputers have also become more computationally powerful and cost has dropped significantly, ushering in the development of distributed computers. The commercial transport industry's interest is shifting away from the task of creating highly reliable systems (now achievable) toward highly available distributed systems to reduce maintenance costs and to garner greater computational resources. Such systems may need to tolerate more than two near-coincident faults, and the automatic HARP near-coincident model may become too conservative. Two options are available. The user can edit the HARP generated ASCII files to correct the next fault rates to the exact ones in files *.ALL, *.SAM, or *.USR (as appropriate), and if necessary, edit the *.MAT files to add additional state transitions as necessary. An exact Markov chain model can be obtained in this manner. An alternative is to use the extended behavioral decomposition multifault model implemented in X-Window System HARP (XHARP) (ref. 5). Volume 1 of this Technical Paper provides more details.

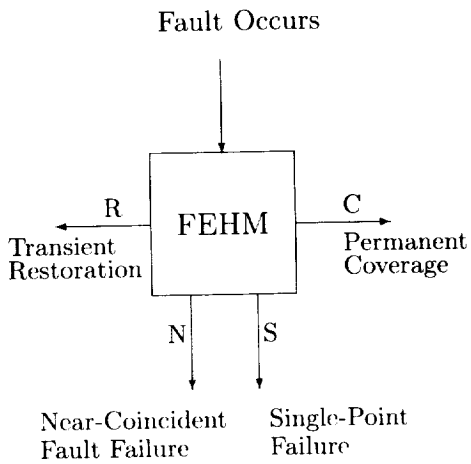


Figure 19. Structure of HARP FEHM.

The coverage model is still solved in isolation; not only the probability of reaching the R, C, and S exits but also the *time* to reach each exit are calculated. If we know (probabilistically) when a near-coincident fault occurs and also the time to reach the R, C, and S exits, we can determine whether one of these exits is reached before the near-coincident fault. A fourth exit N is added to the coverage model leading to a new failure state labeled FNCF (failure near-coincident fault). (See fig. 19.) The probabilities r, c, and s are now adjusted since the exits must be reached before a certain time. Therefore, $N = (1 - C - R - S)$. (See vol. 1 of this TP for the derivation of C, R, and S.)

Again, we automatically incorporate the possibility of imperfect coverage into the perfect coverage Markov chain, as subsequently shown in our three-processor example. Unlike the previous figures of the three-processor system, the FEHM's here have four exits. Note, too, that the exit probabilities are now distinct for FEHM 1 and FEHM 2 (fig. 20) because the next fault rates are state dependent.

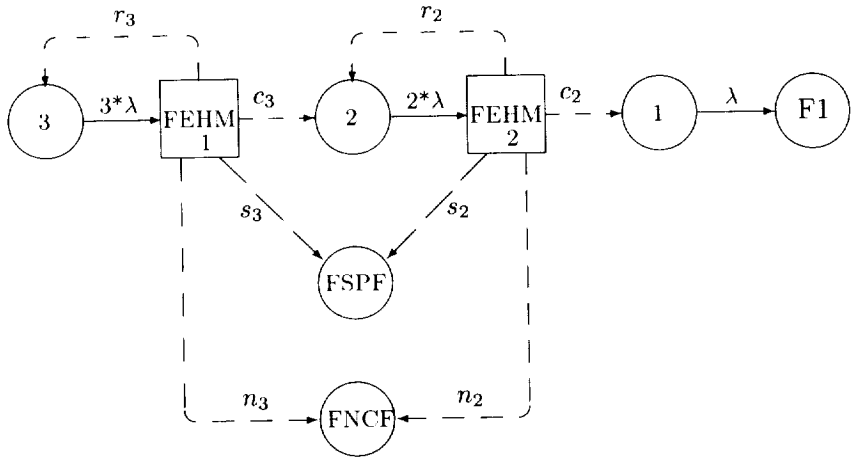


Figure 20. Three-processor system showing FEHM's with C, S, R, and N exit probabilities.

While in the coverage model denoted by FEHM 1, a second processor fault is possible with rate $2*\lambda$. Therefore, one of the exits, R, C, or S must be reached before time to the second fault

(which is an exponentially distributed random variable with parameter $2*\lambda$) if a near-coincident fault is to be avoided. Likewise, while in the coverage model denoted by FEHM 2, another processor failure can occur with rate λ.

Assume that FEHM 1 and FEHM 2 in figure 20 are exponentially distributed delays with rate δ (see fig. 21). Thus, $s = r = 0$. Note that in the absence of a near-coincident fault, $c = 1$. However, with the near-coincident fault occurring at the rate $2 * \lambda$ from FEHM 1, the probability of a successful C exit before the occurrence of a second near-coincident fault is easily shown to be $c_3 = \frac{\delta}{\delta+2*\lambda}$.

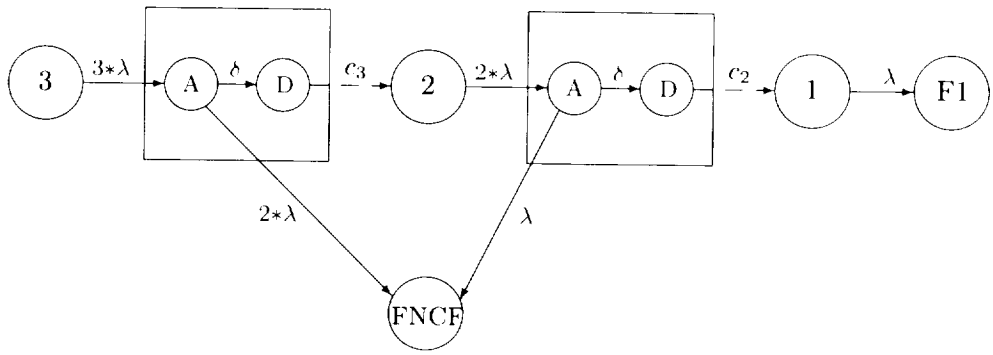


Figure 21. Three-processor system showing near-coincident faults.

Similarly for FEHM 2, $c_2 = \frac{\delta}{\delta+\lambda}$. The instantaneous jump model is shown in figure 21.

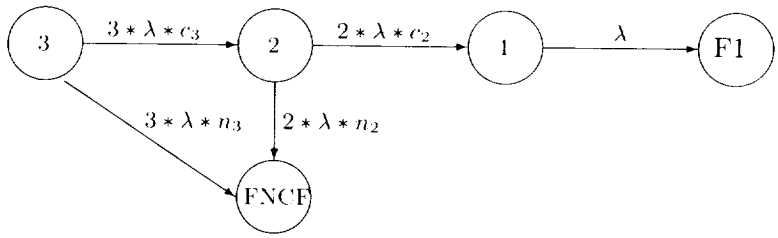


Figure 22. Instantaneous jump model of three-processor system with near-coincident faults.

In figure 22, $n_3 = 1 - c_3$ and $n_2 = 1 - c_2$. Thus, the inclusion of near-coincident faults causes the coverage values to become state dependent. The HARP program automatically derives the coverage factors by taking the Laplace transform of the time-to-exit distributions. We compute the transforms for the single fault model and then substitute the second near-coincident fault rate for the Laplace transform variable to obtain the state-dependent coverage values. If the time-to-exit distribution is not available in closed form, a Taylor series expansion of the Laplace transform yields an expression that depends on powers of the next fault rate and on the moments of the distribution. These moments are easily obtained from empirical or simulation data. See reference 6 for the mathematical derivations.

We need not restrict ourselves to single-state FEHM's. Let us again look at a portion of the CARE III coverage model that was introduced in chapter 4 on permanent faults. While essentially the same model as figure 13, the instantaneous transition labeled P_A in figure 13 is now an instantaneous transition out of the FEHM. (See fig. 23.) We have also added the near-coincident fault rates.

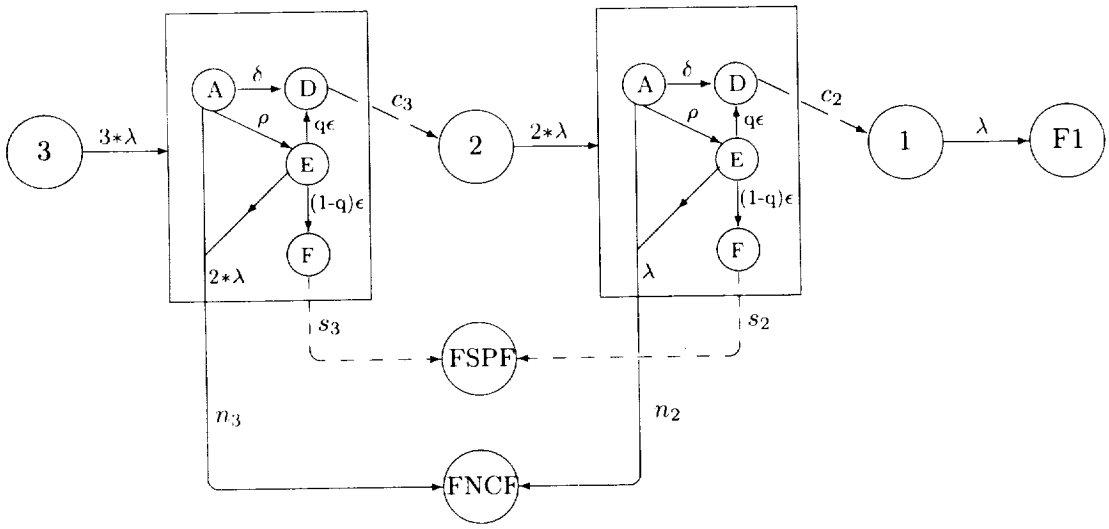


Figure 23. Permanent CARE III FEHM with N, C, and S exits.

Now the FEHM probabilities, when replaced by a branch point are

$$c_3 = \frac{\delta}{\delta+\rho+2\lambda} + \left(\frac{\rho}{\delta+\rho+2\lambda}\right)\left(\frac{q\epsilon}{\epsilon+2\lambda}\right) \qquad c_2 = \frac{\delta}{\delta+\rho+\lambda} + \left(\frac{\rho}{\delta+\rho+\lambda}\right)\left(\frac{q\epsilon}{\epsilon+\lambda}\right)$$

and

$$s_3 = \left(\frac{\rho}{\delta+\rho+2\lambda}\right)\frac{(1-q)\epsilon}{\epsilon+2\lambda} \qquad s_2 = \left(\frac{\rho}{\delta+\rho+\lambda}\right)\frac{(1-q)\epsilon}{\epsilon+\lambda}$$

As before, these probabilities are determined by the HARP program based on the user inputs for the rates and probabilities in the model.

9.2. Near-Coincident Fault Options

As discussed in volume 1 of this Technical Paper, the HARP user has three options (three multifault models) for modeling near-coincident faults. To better demonstrate the various options allowed in HARP, the following Markov model is utilized. In this example, we show that the reduced model after each FEHM has been reduced to a branch point. The arcs entering the FNCF (not shown) are part of the inherent structure of the model. For each C^*, there is a corresponding N^* into the FNCF state.

9.2.1. ALL-Inclusive Near-Coincident Multifault Model

This specification for the interfering fault assumes that a second near-coincident fault *anywhere in the system* (while attempting to handle a first fault) causes immediate system failure (via the FNCF state). The use of this model always gives a conservative result for practical systems of interest. Volume 1 chapter 7 of this Technical Paper presents an example system where the ALL model is specified for a system that has nearly independent fault containment regions. Under certain conditions, the degree of conservative error can be quantified by using HARP's simple lower bound (see vol. 1 of this TP). Another alternative is to modify the HARP generated ASCII files (*.ALL, *.SAM, *.USR and perhaps the *.MAT in some cases) with a text editor for specifying the exact next fault rate(s). When fault rates are specified correctly, an accurate result can be obtained. The simple bounds also become valid for all Markovian models and can be used to gauge the results. Using XHARP (ref. 15) is another alternative.

Given the Markov chain of figure 24, for the arc labeled with a "C1", the next fault rate is $5\lambda_1 + 3\lambda_2 + 2\lambda_3$. This rate is found by looking at the target state (state 2) and taking the maximum of the sum of the incoming arcs minus 1.0 for this component type (λ_1) and the sum of the same component type parameters exiting the state. In addition, the maximum between the incoming and outgoing arcs for every other component type in the dictionary is added to the rate. (However, the dictionary must be complete for a correct rate.) For this system, the all inclusive rate file appears as follows. (Note, the expression following each Ci is the next fault rate corresponding to the Ci transition and is not the coverage value.)

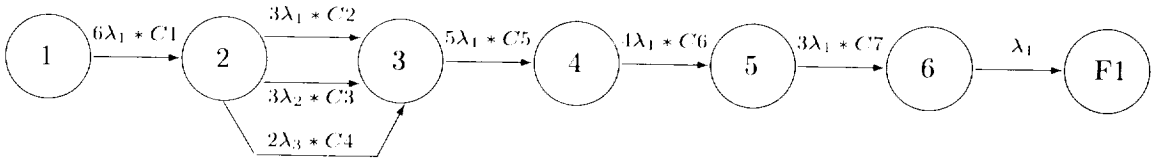


Figure 24. System model for example 1.

```
  FIGURE 24 .ALL file
C1
  5*LAMBDA1+3*LAMBDA2+2*LAMBDA3;
C2
  5*LAMBDA1+3*LAMBDA2+2*LAMBDA3;
C3
  5*LAMBDA1+2*LAMBDA2+2*LAMBDA3;
C4
  5*LAMBDA1+3*LAMBDA2+LAMBDA3;
C5
  4*LAMBDA1;
C6
  3*LAMBDA1;
C7
  2*LAMBDA1;
```

9.2.2. SAME-Type Near-Coincident Multifault Model

More optimistically, the user can assume that only near-coincident faults of the same component type cause system failure (while attempting to handle a first fault). For the FEHM associated with C1 in figure 24, only those components that fail with rate λ_1 cause system failure. The same type files appears as follows:

```
  FIGURE 24 .SAM file
C1
  5*LAMBDA1;
C2
```

```
  5*LAMBDA1;
C3
  2*LAMBDA2;
C4
  LAMBDA3;
C5
  4*LAMBDA1;
C6
  3*LAMBDA1;
C7
  2*LAMBDA1;
```

9.2.3. USER-Defined Near-Coincident Multifault Model

For some models, the user can define explicitly for each component, which components (itself and/or others) can interfere with fault recovery. Thus, the next fault rate for the FEHM's between operational states depends on user input. Let us refer to those components with rate λ_1 as processor1, λ_2 as processor2, and λ_3 as processor3. For this example, the user can specify that all three processors interfere with recovery in the processor1 components, but only processor2 affects recovery in processor2 and only processor3 interferes with its own recovery. While processor1 can be modeled with the all-inclusive fault type and processor2 and processor3 with the same-type next fault rate, only one near-coincident fault rate type can be specified for the entire model. This file appears as follows:

```
  FIGURE 24 .USR file
C1
  5*LAMBDA1+3*LAMBDA2+2*LAMBDA3;
C2
  5*LAMBDA1+3*LAMBDA2+2*LAMBDA3;
C3
  2*LAMBDA2;
C4
  LAMBDA3;
C5
  4*LAMBDA1;
C6
  3*LAMBDA1;
C7
  2*LAMBDA1;
```

9.2.4. Example for .ALL and .SAM Options

Now let $\lambda_2 = \lambda_1$, allowing the user the ability to utilize an overriding FEHM file option. We now remove the λ_3 arc for simplicity. (See fig. 25.)

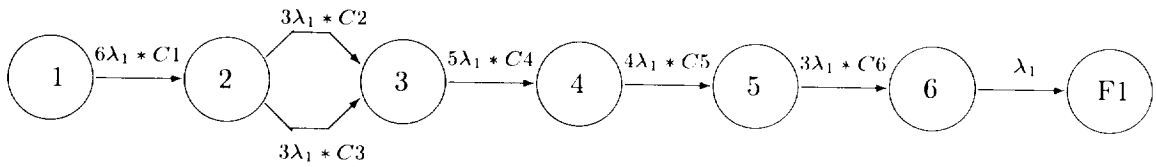


Figure 25. System model for example 2.

For this system, the all inclusive rates are the same as the same type rates.

```
C1
  6*LAMBDA1;
C2
  5*LAMBDA1;
C3
  5*LAMBDA1;
C4
  4*LAMBDA1;
C5
  3*LAMBDA1;
C6
  2*LAMBDA1;
```

9.2.5. No Near-Coincident Faults

If the user chooses, near-coincident faults can be ignored. As a result, the probability of being in state FNCF is zero.

9.3. Specification of Near-Coincident Fault Rates

The user need not worry about the actual near-coincident fault rates in HARP. While running the engine program (*harpeng*), the user is asked which near-coincident fault rate to use. Once one of the four options is chosen (ALL-inclusive, SAME-type, USER-defined, NONE), the program automatically determines the correct next fault rate. When no coverage models were specified in the *tdrive* program or no near-coincident faults were specified in the *fiface* program, the user is not asked about near-coincident faults in *harpeng*.

Chapter 10

Error Bounds

This section addresses the question posed by the engine pertaining to running bounds. In general, the *tdrive* and *fiface* programs process a system model using failure rates that are symbolic rather than numeric (of course, numeric values can be entered in the Markov chain in *tdrive* if desired). The user binds the numeric values to the symbolic rates during the execution of *harpeng*. Because many input parameters to the FORM model are not known exactly (i.e., coverage values from simulation are given as confidence intervals and the user may only know a range of values for the failure rates), HARP enables the user to specify the rates as a nominal value plus or minus a variation.

Two different kinds of bounds are provided by the HARP program, simple model (parametric) bounds and model truncation bounds. Depending on the system being modeled, none, one, or both kinds of bounds may be applicable.

The simple parametric bounds are computed for two distinct classes of models: The AS IS model that does not use any FEHM's and does not invoke behavioral decomposition, and those models that do invoke FEHM's and behavioral decomposition. Both model classes can also be modified to reflect the model state reduction technique called truncation. (See vol. 1 of this TP.)

The AS IS model is used strictly for parametric analysis that reports the effect of system unreliability as a function of the user-specified parametric variation. These data are useful for sensitivity analyses. The simple parametric bounds for this model class are true bounds for the original user-specified model. (See vol. 1 of this TP.)

When FEHM's and behavioral decomposition are invoked, the simple bounds take on two manifestations. When no parametric variation is specified and the user selects the simple bounds computation (prompted by HARP), simple upper and lower bounds are computed based on estimated maximum and minimum imperfect coverage and lack of sufficient redundancy. If in addition, parametric variation is specified, a combined effect is estimated, that is, imperfect coverage with insufficient redundancy and parametric variation. Unlike the AS IS model, the simple lower bound associated with behavioral decomposition is a conditional bound. When many fault containment regions are modeled, the lower bound may not bound the full model unreliability but will bound the HARP instantaneous jump model unreliability. (See vol. 1 of this TP.)

HARP does not allow bounds to be evaluated when any failure rate is Weibull. When the system being modeled has repair, bounds are evaluated only when an absorbing state is present in the model.

10.1. Simple Model (Parametric) Bounds

10.1.1. AS IS Model

Because many input parameters to the FORM model are not known exactly (e.g., the user may only know a range of values for the failure rates), HARP allows the FORM input parameters to be expressed in terms of ranges of values rather than point estimates. HARP produces upper and lower bounds on the system unreliability that are a function of these ranges of values. The model evaluates the overall system failure probability by taking the lower bound on the failure rates and the upper bound on the repair rates as the best case and by taking the upper bound

on the failure rates and the lower bound on the repair rates as the worst case. The model also produces the predicted unreliability based on the nominal values. The simple parametric bounds for this model class are bounds for the original user-specified model.

10.1.2. Models With Behavioral Decomposition

We approach the analysis of errors by decomposing the original model into two simpler models that can be combined to obtain a conservative unreliability estimate (refs. 19 to 21). The general form of the simple bounds is given as:

$$P\left(A \bigcup B\right) \leq \min\left[1, P\left(A_{\text{high}}\right) + P(B_{\max})\right]$$

$$P\left(A \bigcup B\right) \geq \max[P(A_{\text{low}}), P(B_{\min})]$$

The first rule gives the conservative bound, and the second rule gives the optimistic bound[10].

The first expression gives the upper unreliability bound, and the second gives the lower unreliability bound. $P(A)$ is the system failure probability caused by the lack of sufficient redundancy. $P(A_{\text{high}})$ and $P(A_{\text{low}})$ are used instead of $P(A)$ when parametric tolerance is selected to cause $P(A)$ to be maximum to get $P(A_{\text{high}})$ and to cause $P(A)$ to be minimum to get $P(A_{\text{low}})$. $P(B)$ is the probability of system failure due to imperfect coverage. When FEHM's are specified for behavioral decomposition, $P(B)$ is computed for the minimum imperfect coverage to get $P(B_{\min})$ and the maximum imperfect coverage to get $P(B_{\max})$. $P(A)$ is further modified when transients are specified in at least one FEHM. The perfect redundancy model (coverage assumed to be perfect) transition rates are modified by coefficients that reflect transient restoration probabilities. The net effect is to reduce the probability of failure by redundancy exhaustion since transient restoration occurs.

The simple bounds computed by HARP are the bounds on the instantaneous coverage model (see vol. 1 of this TP) that produces the unreliability result and also bounds the user's full model under certain conditions: The simple upper bound on the system unreliability is always a true bound with respect to both the instantaneous coverage model and the user-specified model (provided all failure rates are constant).[11]

The validity of the optimistic lower bound with respect to the user-specified model is dependent on the use of large numbers of fault containment regions that require the ALL multifault model (see chapters 1 and 7 of this TP for details).

The HARP simple bounds are used for preliminary estimates of unreliability. They are provided as a quick-look computation that can be used in the early stages of system design when only ranges of parameter values are available. The essence of HARP output is the nominal result (instantaneous jump model unreliability) and not the simple bounds. If the model is solved AS IS, without any FEHM's or with the VALUES FEHM, the HARP bounds are true bounds for the user-specified full model. With FEHM's, the upper bound is always a true bound, and the lower bound is also a true bound except possibly for a limited class of systems with many fault containment regions.

[10] Validity of these bounds is subject to the correct specification of multifault models, where applicable (see vol. 1 of this TP).

[11] HARP FEHM's and multifault models only support single recovery transitions. System models with multiple recovery transitions can cause the simple upper bound to improperly bound the HARP unreliability result or the full model. For such systems, the user can edit HARP-generated ASCII files with a text editor to specify the correct model. In this case, the bounds will be valid. XHARP provides another modeling alternative. The HARP AS IS model can also be used to provide accurate results.

10.2. Truncation Bounds

Truncation bounds are obtained as follows. When the truncated model is solved, the probability of being in each of the TA states is calculated. By adding these probabilities to that of the DOWN (failed) states (DS) before the truncation line, we get an upper bound on the system unreliability (SU). This result assumes that all states beyond the truncation line are failed states. To get a lower bound on unreliability, we add only the probabilities of the failure states before the truncation line. In this case, the TA states are automatically considered to be functional states by HARP. To use some notation, the states in the truncated model are denoted with a subscript tr and the states in the full model have the subscript full. The bounds on the system unreliability are given by the following equation:

$$Pr(DS_{\text{tr}}) \leq SU_{\text{full}} \leq Pr(TA_{\text{tr}}) + Pr(DS_{\text{tr}})$$

HARP not only gives the system unreliability but also provides a breakdown in terms of individual failure probabilities. Failure causes are the exhaustion of different components, FNCF and FSPF. In a truncated model, HARP gives bounds on the system unreliability as well as individual failure probabilities. F1 denotes a state where fewer than the minimum required of component type 1 are still operational. If there is an F1 state before the truncation level, we use the probability of being in the F1 state as a lower bound on the probability of failure due to exhaustion of component 1. All transitions due to failure of component 1 that fall on the truncation line and do not lead to state F1 are directed into a state called TA1.

Probability of failure due to exhaustion of component 1, $Pr(F1_{\text{full}})$, is bounded as follows:

$$Pr(F1_{\text{tr}}) \leq Pr(F1_{\text{full}}) \leq Pr(TA1_{\text{tr}}) + Pr(F1_{\text{tr}})$$

The bounds on the probability of exhaustion of other components are obtained in a similar manner. Now we obtain bounds for the probability of a near-coincident fault and a single-point fault.

The probability of being in the FNCF state before the truncation level is a lower bound on the FNCF probability. The upper bound is taken to be this lower bound probability added to the combined probability of all TA states:

$$Pr(FNCF_{\text{tr}}) \leq Pr(FNCF_{\text{full}}) \leq Pr(TA_{\text{tr}}) + Pr(FNCF_{\text{tr}})$$

The bounds on probability of single-point fault are obtained in a similar manner as given as follows:

$$Pr(FSPF_{\text{tr}}) \leq Pr(FSPF_{\text{full}}) \leq Pr(TA_{\text{tr}}) + Pr(FSPF_{\text{tr}})$$

10.3. Combined Bounds

When parametric bounds (via a simple model) are desired from a truncated model, the bounds are combined in the following way. The simple model solution uses the optimistic parameters (lowest possible failure rates, highest possible repair rates and coverage factors) to produce an upper bound on the reliability of the system (ref. 20).

$$R_{\text{high}}(t) = 1 - \max[P_{\text{eshlow}}(t), P_{\text{covlow}}(t)]$$

If the model from which the simple bounds are derived is a truncated model, then the TA states are taken to be operational states (for the optimistic bound). Likewise, the simple

model solution uses the pessimistic parameters (highest possible failure rates, lowest possible coverage factors and repair rates) to produce a lower bound on the unreliability of the system (ref. 20).

$$R_{\text{low}}(t) = 1 - \min\left[P_{\text{eshhigh}}(t) + P_{\text{covhigh}}(t), 1\right]$$

If the model from which the simple model bounds are derived is a truncated model, then the TA states are taken to be failure states (for the pessimistic bounds). The first type of bounds are reported as "simple model bounds," the second type are reported as "truncated model bounds," and the combined bounds are reported as "truncated simple model bounds."

The use of behavioral decomposition and instantaneous coverage factors have been proven to result in conservative estimates of reliability (ref. 22), when failure rates are constant (exponential times to failure). Both bounding techniques (simple and truncation) produce bounds on this conservative estimate of reliability. For the class of practical highly reliable systems, the HARP (simple and truncation) bounds also encompass the reliability of the original model.

If a model is extremely large and cannot fit in the engine data structure simultaneously, the bounds are disallowed. Also, if a model has Weibull failure rates or no absorbing states, bounds are not asked for nor provided.

NASA Langley Research Center
Hampton, VA 23681-0001
June 15, 1994

Appendix A

File Listings of HARP Examples

A1. Example 3PFT1

Example 3PFT1 presents a model of a three processor (triplex) system that was input as a fault tree. Each component is specified as a unique basic event. (See section 3.1.)

File 3PFT1.DIC gives the output from program *tdrive*.

```
 1 PROCESSOR1     LAMBDA1        NONE
   INTERFERING COMPONENT TYPES:
 2 PROCESSOR2     LAMBDA2        NONE
   INTERFERING COMPONENT TYPES:
 3 PROCESSOR3     LAMBDA3        NONE
   INTERFERING COMPONENT TYPES:
FEIDS
  10  9  8
```

File 3PFT1.INT gives the output from program *tdrive*.

```
SORTED
1     2        LAMBDA1;
1     3        LAMBDA2;
1     4        LAMBDA3;
2     5        LAMBDA2;
2     6        LAMBDA3;
3     5        LAMBDA1;
3     7        LAMBDA3;
4     6        LAMBDA1;
4     7        LAMBDA2;
5     8        LAMBDA3*X;
6     9        LAMBDA2*X;
7     10       LAMBDA1*X;
```

File 3PFT1.MAT gives the output from program *fiface*.

```
 10
 2  ,  1
LAMBDA1;
 3  ,  1
```

```
 LAMBDA2;
  4  ,  1
 LAMBDA3;
  5  ,  2
 LAMBDA2;
  5  ,  3
 LAMBDA1;
  6  ,  2
 LAMBDA3;
  6  ,  4
 LAMBDA1;
  7  ,  3
 LAMBDA3;
  7  ,  4
 LAMBDA2;
  8  ,  5
 LAMBDA3*X;
  9  ,  6
 LAMBDA2*X;
  10  ,  7
 LAMBDA1*X;
  0,0
```

File 3PFT1.SYM gives the output from program *fiface*.

```
  X
  999
END SYMBOL DEFINITION
F1
  1010
F2
  1009
F3
  1008
END FAILURE STATE DEFINITION
```

File 3PFT1.RS1 gives the output from program *harpeng*.

```
                ---------- HARP ----------
                - The Hybrid Automated Reliability Predictor -
                ---------- Release Version 7.0 ----------
                ---------- February 1993 ----------

Modelname:
    3PFT1
 Input description (from dictionary file):
 Component type: 1      Name: PROCESSOR1
 Symbolic failure rate:
 LAMBDA1      Constant failure rate:
                    0.10000000D-02      +/-       0.00000000D+00
 FEHM file name: NONE
 Component type: 2      Name: PROCESSOR2
 Symbolic failure rate:
 LAMBDA2      Constant failure rate:
                    0.10000000D-02      +/-       0.00000000D+00
 FEHM file name: NONE
 Component type: 3      Name: PROCESSOR3
 Symbolic failure rate:
 LAMBDA3      Constant failure rate:
                    0.10000000D-02      +/-       0.00000000D+00
 FEHM file name: NONE
 NO near-coincident faults considered.
 Time(in Hours):    0.100D+02
 State Probabilities
 State name: F1            0.32837475D-06
 State name: F2            0.32837475D-06
 State name: F3            0.32837475D-06
              ------------------------------------------------
                Reliability =   0.99999901D+00
              Unreliability =   0.98512425D-06
  Total failure by redundancy exhaustion =   0.98512425D-06
  GERK ODE solver:  global error value   0.107D-16
                  relative error value   0.100D-08
```

```
See Users Guide, section 3.3 for interpretation.

      0 Reports from the GERK ODE solver.
```

A2. Example 3PFT2

Example 3PFT2 is identical to the previous one except that a replication factor of three is specified for the basic events to aggregate the unique basic events into one basic event. (See section 3.2.)

File 3PFT2.DIC gives the output from program *tdrive*.

```
  1 PROCESSOR     LAMBDA        NONE

    INTERFERING COMPONENT TYPES:

FEIDS

  4
```

File 3PFT2.INT gives the output from program *tdrive*.

```
SORTED

  1     2        3*LAMBDA;

  2     3        2*LAMBDA;

  3     4        LAMBDA*X;
```

File 3PFT2.MAT gives the output from program *fiface*.

```
  4

  2  ,  1

 3*LAMBDA;

  3  ,  2

 2*LAMBDA;

  4  ,  3

 LAMBDA*X;

  0,0
```

File 3PFT2.SYM gives the output from program *fiface*.

```
  X

  999

END SYMBOL DEFINITION

F1

  1004

END FAILURE STATE DEFINITION
```

File 3PFT2.RS1 gives the output from program *harpeng.*

```
                ---------------- HARP ----------------------
                - The Hybrid Automated Reliability Predictor -
                ------------ Release Version 7.0 -------------
                --------------- February 1993 ---------------
Modelname:
    3PFT2
 Input description (from dictionary file):
 Component type: 1      Name: PROCESSOR
 Symbolic failure rate:
 LAMBDA       Constant failure rate:
                    0.10000000D-02     +/-       0.00000000D+00
 FEHM file name: NONE
 NO near-coincident faults considered.
 Time(in Hours):    0.100D+02
 State Probabilities
 State name: F1             0.98512425D-06
             ------------------------------------------------
               Reliability =   0.99999901D+00
             Unreliability =   0.98512425D-06
 Total failure by redundancy exhaustion =   0.98512425D-06
 GERK ODE solver:  global error value   0.716D-17
                  relative error value   0.100D-08
 See Users Guide, section 3.3 for interpretation.
       0 Reports from the GERK ODE solver.
```

A3. Example 3PMC

Example 3PMC is the same system model as previously displayed, but it is entered as a Markov chain directly in lieu of a fault tree. (See section 3.3.)

File 3PMC.DIC gives the output from program *tdrive.*

```
  -1 ASIS
```

File 3PMC.INT gives the output from program *tdrive.*

```
UNSORTED
3   2  3*LAMBDA;
2  1  2*LAMBDA;
1  F1 LAMBDA;
```

File 3PMC.MAT gives the output from program *fiface*.

```
 4
 2  ,  1
3*LAMBDA;
 3  ,  2
2*LAMBDA;
 4  ,  3
LAMBDA*X;
 0,0
```

File 3PMC.SYM gives the output from program *fiface*.

```
 X
 999
END SYMBOL DEFINITION
F1
 1004
END FAILURE STATE DEFINITION
```

File 3PMC.RS1 gives the output from program *harpeng*.

```
                ----------------- HARP ----------------------
                - The Hybrid Automated Reliability Predictor -
                ------------ Release Version 7.0 -------------
                --------------- February 1993 ----------------
Modelname:
    3PMC
 Symbolic values:
 Symbolic failure rate:
 LAMBDA       Constant failure rate:
                     0.10000000D-02      +/-       0.00000000D+00
 NO near-coincident faults considered.
 Time(in Hours):    0.100D+02
 State Probabilities
 State name: F1             0.98512425D-06
              -------------------------------------------------
                Reliability =   0.99999901D+00
               Unreliability =   0.98512425D-06
 Total failure by redundancy exhaustion =   0.98512425D-06
```

```
GERK ODE solver:  global error value    0.716D-17
                  relative error value   0.100D-08
See Users Guide, section 3.3 for interpretation.
       0 Reports from the GERK ODE solver.
```

A4. Example 3PCARE1

These files are for the replicated triplex processor system with a CARE FEHM specified for permanent faults. (See section 6.)

FILE 3PCARE1.DIC output from program *tdrive*.

```
 1 PROCESSOR      LAMBDA          CARE1.FHM
   INTERFERING COMPONENT TYPES:
FEIDS
4
```

File 3PCARE1.INT gives the output from program *tdrive*.

```
SORTED
 1   2    3*LAMBDA;
 2   3    2*LAMBDA;
 3   4    LAMBDA*X;
```

FEHM FILE CARE1.FHM

```
    CARE.SINGLE.FAULT.MODEL
   PROBABILITY OF PERMANENT: 0.10000000d+01
   PROBABILITY OF INTERMITTENT: 0.00000000d+00
   PROBABILITY OF TRANSIENT: 0.00000000d+00
    PERMANENT MODEL PARAMETERS
    --------------------------
   DELTA:  0.36000000d+03
   EPSILON:  0.36000000d+04
   RHO:  0.18000000d+03
   PA:  0.10000000d+01
   Q:  0.99900000d+00
```

File 3PCARE1.MAT gives the output from program *fiface*.

```
 5
 2  ,  1
3*LAMBDA*C1;
 3  ,  2
2*LAMBDA*C1;
```

```
 4  ,  3
LAMBDA*X;
 5  ,  1
3*LAMBDA*S1;
 5  ,  2
2*LAMBDA*S1;
 0,0
```

File 3PCARE1.SYM gives the output from program *fiface*.

```
C1
3
CARE1.FHM
  X
  999
END SYMBOL DEFINITION
F1
  1004
FSPF
  1005
END FAILURE STATE DEFINITION
```

File 3PCARE1.RS1 gives the output from program *harpeng*.

```
                ----------------- HARP ----------------------
                - The Hybrid Automated Reliability Predictor -
                ------------ Release Version 7.0 -------------
                --------------- February 1993 ----------------
Modelname:
    3PCARE1
 Input description (from dictionary file):
 Component type: 1      Name: PROCESSOR
 Symbolic failure rate:
 LAMBDA       Constant failure rate:
                    0.10000000D-02     +/-       0.00000000D+00
 FEHM file name: CARE1.FHM
     For this FEHM model, the exit probabilities are:
        (in the absence of near-coincident faults)
          Transient restoration:       0.00000000D+00
```

```
        Permanent coverage:          0.99966667D+00
        Single-point failure:        0.33333333D-03
NO near-coincident faults considered.
Time(in Hours):    0.100D+02
State Probabilities
State name: F1             0.98446761D-06
State name: FSPF           0.99498051D-05
-------------------------------------------------
              Reliability =   0.99998907D+00
            Unreliability =   0.10934273D-04
Total failure by redundancy exhaustion =   0.98446761D-06
GERK ODE solver:  global error value   0.107D-16
                  relative error value   0.100D-08
See Users Guide, section 3.3 for interpretation.
       0 Reports from the GERK ODE solver.
```

A5. Example 3PARIES

These files are for the replicated triplex processor system with an ARIES FEHM specified. (See section 6.2).

FEHM file ARIES.FHM

```
 ARIES.TRANSIENT.RECOVERY.MODEL
PROBABILITY THAT FAULT IS TRANSIENT 0.90000000d+00
MEAN DURATION OF TRANSIENT FAULT 0.50000000d-02
PROBABILITY THAT FAULT IS CATASTROPHIC 0.10000000d-02
NUMBER OF TRANSIENT RECOVERY PHASES       3
  PHASE 1 DURATION:  0.80000000d+00 EFFECTIVENESS:  0.80000000d+00
  PHASE 2 DURATION:  0.20000000d+00 EFFECTIVENESS:  0.70000000d+00
  PHASE 3 DURATION:  0.10000000d+00 EFFECTIVENESS:  0.50000000d+00
  FAILURE RATE OF RECOVERY SYSTEM HARDWARE:  .00000000D+00
COVERAGE OF PERMANENT FAULT:  0.85000000d+00
```

File 3PARIES.RS1 gives the output from program *harpeng*.

```
                ---------------- HARP ----------------------
                - The Hybrid Automated Reliability Predictor -
                ------------ Release Version 7.0 -------------
                -------------- February 1993 ----------------
Modelname:
```

```
   3PARIES1
Input description (from dictionary file):
Component type: 1      Name: PROCESSOR
Symbolic failure rate:
LAMBDA       Constant failure rate:
                  0.10000000D-02     +/-       0.00000000D+00
FEHM file name: ARIES.FHM
    For this FEHM model, the exit probabilities are:
       (in the absence of near-coincident faults)
         Transient restoration:       0.76423498D+00
         Permanent coverage:          0.199559027D+00
         Single-point failure:        0.36214753D-01
NO near-coincident faults considered.
Time(in Hours):    0.100D+02
State Probabilities
State name: F1             0.99891519D+00
State name: FSPF           0.10848086D-02
-------------------------------------------------
              Reliability =   0.99891519D+00
            Unreliability =   0.10848086D-02
Total failure by redundancy exhaustion =   0.39604048D-07
GERK ODE solver:  global error value   0.280D-19 {Depend on computing}
                 relative error value   0.100D-08 {platform}
See Users Guide, section 3.3 for interpretation.
      0 Reports from the GERK ODE solver.
```

A6. Example 3PCARE2

These files are for the triplex processor system using the CARE FEHM with transient faults. (See section 6.4.)

FEHM FILE CARE2.FHM

```
    CARE.SINGLE.FAULT.MODEL
   PROBABILITY OF PERMANENT: 0.00000000d+00
   PROBABILITY OF INTERMITTENT: 0.00000000d+00
   PROBABILITY OF TRANSIENT: 0.10000000d+01
    TRANSIENT MODEL PARAMETERS
    -------------------------
```

```
ALPHA:  0.36000000d+05
DELTA:  0.36000000d+03
EPSILON:  0.36000000d+04
RHO:  0.18000000d+03
PA:  0.50000000d+00
PB:  0.50000000d+00
Q:  0.90000000d+00
```

File 3PCARE2.RS1 gives the output from program *harpeng*.

```
                ----------------- HARP ----------------------
                - The Hybrid Automated Reliability Predictor -
                ------------ Release Version 7.0 -------------
                --------------- February 1993 ----------------
Modelname:
    3PCARE2
 Input description (from dictionary file):
 Component type: 1      Name: PROCESSOR
 Symbolic failure rate:
 LAMBDA       Constant failure rate:
                    0.10000000D-02     +/-       0.00000000D+00
 FEHM file name: CARE2.FHM
     For this FEHM model, the exit probabilities are:
        (in the absence of near-coincident faults)
          Transient restoration:       0.99232518D+00
          Permanent coverage:          0.71796719D-02
          Single-point failure:        0.49514978D-03
 NO near-coincident faults considered.
 Time(in Hours):    0.100D+02
 State Probabilities
 State name: F1             0.51414141D-10
 State name: FSPF           0.14853850D-04
              ------------------------------------------------
                Reliability =   0.99998515D+00
              Unreliability =   0.14853901D-04
 Total failure by redundancy exhaustion =   0.51414141D-10
 GERK ODE solver:  global error value   0.358D-17
```

```
                relative error value   0.100D-08
See Users Guide, section 3.3 for interpretation.
      0 Reports from the GERK ODE solver.
```

A7. Example 3PCARE3

These files are for the triplex processor using the CARE FEHM with intermittent faults. (See section 7.)

FEHM file CARE3.FHM

```
 CARE.SINGLE.FAULT.MODEL
PROBABILITY OF PERMANENT: 0.20000000d+00
PROBABILITY OF INTERMITTENT: 0.20000000d+00
PROBABILITY OF TRANSIENT: 0.60000000d+00
 PERMANENT MODEL PARAMETERS
 -------------------------
DELTA:  0.30000000d+03
EPSILON:  0.36000000d+04
RHO:  0.24000000d+03
PA:  0.10000000d+01
Q:  0.99900000d+00
 INTERMITTENT MODEL PARAMETERS
 ----------------------------
ALPHA:  0.21000000d+04
BETA:  0.30000000d+04
DELTA:  0.36000000d+03
EPSILON:  0.36000000d+04
RHO:  0.18000000d+03
PA:  0.90000000d+00
PB:  0.10000000d+00
Q:  0.99900000d+00
 TRANSIENT MODEL PARAMETERS
 -------------------------
ALPHA:  0.36000000d+05
DELTA:  0.18000000d+03
EPSILON:  0.36000000d+04
RHO:  0.18000000d+03
PA:  0.50000000d+00
PB:  0.50000000d+00
Q:  0.99900000d+00
```

File 3PCARE3.RS1 gives the output from program *harpeng*.

```
                ----------------- HARP ----------------------
                - The Hybrid Automated Reliability Predictor -
                ------------ Release Version 7.0 -------------
                --------------- February 1993 ----------------
Modelname:
    3PCARE3
 Input description (from dictionary file):
 Component type: 1       Name: PROCESSOR
 Symbolic failure rate:
 LAMBDA       Constant failure rate:
                    0.10000000D-02      +/-       0.00000000D+00
 FEHM file name: CARE3.FHM
     For this FEHM model, the exit probabilities are:
        (in the absence of near-coincident faults)
          Transient restoration:        0.59702017D+00
          Permanent coverage:           0.40280819D+00
          Single-point failure:         0.17163782D-03
 NO near-coincident faults considered.
 Time(in Hours):    0.100D+02
 State Probabilities
 State name: F1              0.16103642D-06
 State name: FSPF            0.51387370D-05
              ------------------------------------------------
                Reliability =   0.99999470D+00
              Unreliability =   0.52997734D-05
 Total failure by redundancy exhaustion =   0.16103642D-06
 GERK ODE solver:  global error value   0.358D-17
                  relative error value   0.100D-08
 See Users Guide, section 3.3 for interpretation.
        0 Reports from the GERK ODE solver.
```

A8. Example 3MMOMENTS

These files are for the triplex processor system with the Moments FEHM. The Moments FEHM is substituted for the ARIES FEHM. (See section 6.2.)

FEHM file FEHM.MOM

```
PROBABILITIES.AND.MOMENTS
TRANSIENT RESTORATION EXIT:
  EXIT PROBABILITY: .9800
  FIRST MOMENT OF TIME TO EXIT:    0.
  SECOND MOMENT OF TIME TO EXIT:    0.
  THIRD MOMENT OF TIME TO EXIT:    0.
RECONFIGURATION COVERAGE EXIT:
  EXIT PROBABILITY: .1615e-01
  FIRST MOMENT OF TIME TO EXIT: 45.00
  SECOND MOMENT OF TIME TO EXIT: .2500
  THIRD MOMENT OF TIME TO EXIT:    0.
SINGLE POINT FAILURE EXIT:
  EXIT PROBABILITY: .3850e-02
  FIRST MOMENT OF TIME TO EXIT:    0.
  SECOND MOMENT OF TIME TO EXIT:    0.
  THIRD MOMENT OF TIME TO EXIT:    0.
```

File 3MMOM.RS1 gives the output from program *harpeng*.

```
                ---------------- HARP ----------------------
                - The Hybrid Automated Reliability Predictor -
                ------------ Release Version 7.0 ------------
                -------------- February 1993 ---------------
Modelname:
   3MMOM
 Input description (from dictionary file):
 Component type: 1      Name: PROCESSOR
 Symbolic failure rate:
 LAMBDA       Constant failure rate:
                    0.10000000D-02     +/-       0.00000000D+00
 FEHM file name: MOM.FHM
    For this FEHM model, the exit probabilities are:
       (in the absence of near-coincident faults)
```

```
        Transient restoration:          0.98000000D+00
        Permanent coverage:             0.16150000D-01
        Single-point failure:           0.38500000D-02
NO near-coincident faults considered.
Time(in Hours):    0.100D+02
State Probabilities
State name: F1              0.26010668D-09
State name: FSPF            0.11548400D-03
            ------------------------------------------------
              Reliability =   0.99988452D+00
            Unreliability =   0.11548426D-03
Total failure by redundancy exhaustion =   0.26010668D-09
GERK ODE solver:  global error value   0.175D-20
                  relative error value   0.100D-08
See Users Guide, section 3.3 for interpretation.
       0 Reports from the GERK ODE solver.
```

A9. Example 3MDIST

These files are for the triplex processor system with the Distributions and Probabilities FEHM. The Distributions and Probabilities FEHM is substituted for the ARIES FEHM. (See section 6.2.)

FEHM file DIS.FHM

```
DISTRIBUTIONS.AND.PROBABILITIES
TRANSIENT RESTORATION EXIT:
  EXIT PROBABILITY:  0.00000000d+00
RECONFIGURATION COVERAGE EXIT:
  EXIT PROBABILITY:  0.99000000d+00
DISTRIBUTION TYPE: EXP
  RATE:  0.16670000d-01
SINGLE POINT FAILURE EXIT:
  EXIT PROBABILITY:  0.10000000d-01
DISTRIBUTION TYPE: CONSTANT
  VALUE: 0.00000000d+00
```

File 3MDIST.RS1 gives the output from program *harpeng*.

```
                ----------------- HARP ----------------------
                - The Hybrid Automated Reliability Predictor -
                ------------ Release Version 7.0 -------------
                --------------- February 1993 ----------------
Modelname:
    3PDIS
 Input description (from dictionary file):
 Component type: 1       Name: PROCESSOR
 Symbolic failure rate:
 LAMBDA        Constant failure rate:
                     0.10000000D-02      +/-        0.00000000D+00
 FEHM file name: DIS.FHM
     For this FEHM model, the exit probabilities are:
        (in the absence of near-coincident faults)
          Transient restoration:        0.00000000D+00
          Permanent coverage:           0.99000000D+00
          Single-point failure:         0.10000000D-01
NO near-coincident faults considered.
Time(in Hours):     0.100D+02
State Probabilities
State name: F1              0.96552028D-06
State name: FSPF            0.29846563D-03
             ------------------------------------------------
               Reliability =    0.99970057D+00
             Unreliability =    0.29943115D-03
Total failure by redundancy exhaustion =    0.96552028D-06
GERK ODE solver:  global error value    0.358D-17
                  relative error value    0.100D-08
See Users Guide, section 3.3 for interpretation.
       0 Reports from the GERK ODE solver.
```

A10. Example 3P2B

These files are for a triplex processor and dual bus system with the ESPN FEHM for the processors and the VALUES FEHM for the bus. The state probabilities differ from your execution of this example. (See section 8.)

File 3P2B.DIC gives the output from program *tdrive*.

```
 1 PROCESSOR      LAMBDA          ESPN.FHM
   INTERFERING COMPONENT TYPES: 2
 2 BUS            MU              VALUES
   INTERFERING COMPONENT TYPES:
FEIDS
  7  6
```

File 3P2B.INT gives the output from program *tdrive*.

```
SORTED
  1   2    3*LAMBDA;
  1   3    2*MU;
  2   4    2*LAMBDA;
  2   5    2*MU;
  3   5    3*LAMBDA;
  3   6    MU*X;
  4   7    LAMBDA*X;
  4   8    2*MU;
  5   8    2*LAMBDA;
  5   6    MU*X;
  8   7    LAMBDA*X;
  8   6    MU*X;
```

FEHM File ESPN.FHM

```
HARP.SINGLE.FAULT.MODEL
COVERAGE INPUT PARAMETERS:
TIME                          DISTRIBUTION AND PARAMETERS
----                          ---------------------------
ACTIVE TRANSITION             UNIF      0.      1.000
BENIGN TRANSITION             UNIF      0.      .5000
TRANSIENT LIFETIME            EXP     100.0       0.
DETECT TRANSITION             UNIF      0.      .4000
ERROR TRANSITION              WEBUL   10.00     2.500
ERROR-DETECT TRANSITION       WEBUL   50.00     .2500
```

```
ISOLATION TRANSITION                NORML  4.000     1.000
RECOVERY TRANSITION                 ERLNG  100.0     2.000
RECONFIGURATION TRANSITION          NORML  1.000     .5000
OTHER PARAMETERS:
PROBABILITY OF FAULT DETECTION BY SELF TEST: 0.9000
PROBABILITY OF ERROR DETECTION: 0.9000
PROB. OF ISOLATING DETECTED FAULT: 0.9000
NUMBER OF RECOVERY ATTEMPTS:     5
PROB. OF SUCCESSFUL RECONFIGURATION: 0.9000
FRACTION OF FAULTS WHICH ARE TRANSIENT: 0.5000
FRACTION OF FAULTS WHICH ARE PERMANENT: 0.4000
DESIRED CONFIDENCE LEVEL: 90%
ALLOWABLE ERROR: 10%
```

File 3P2B.MAT gives the output from program *fiface*.

```
 10
 2  ,  1
3*LAMBDA*C1;
 3  ,  1
2*MU*C2;
 4  ,  2
2*LAMBDA*C3;
 5  ,  2
2*MU*C2;
 5  ,  3
3*LAMBDA*C4;
 6  ,  3
MU*X;
 6  ,  5
MU*X;
 6  ,  8
MU*X;
 7  ,  4
LAMBDA*X;
 7  ,  8
LAMBDA*X;
```

```
 8   ,   4
2*MU*C2;
 8   ,   5
2*LAMBDA*C5;
 9   ,   1
3*LAMBDA*S1+2*MU*S2;
 9   ,   2
2*LAMBDA*S3+2*MU*S2;
 9   ,   3
3*LAMBDA*S4;
 9   ,   4
2*MU*S2;
 9   ,   5
2*LAMBDA*S5;
 10   ,   1
3*LAMBDA*N1+2*MU*N2;
 10   ,   2
2*LAMBDA*N3+2*MU*N2;
 10   ,   3
3*LAMBDA*N4;
 10   ,   4
2*MU*N2;
 10   ,   5
2*LAMBDA*N5;
 0,0
```

File 3P2B.SYM gives the output from program *fiface*.

```
C1
3
ESPN.FHM
C2
7
  0.500000  0.000000
R2
8
  0.300000  0.000000
```

```
N2
9
  0.000000  0.000000
S2
10
  0.200000  0.000000
C3
3
ESPN.FHM
C4
3
ESPN.FHM
C5
3
ESPN.FHM
  X
  999
END SYMBOL DEFINITION
F1
  1007
F2
  1006
FSPF
  1009
FNCF
  1010
END FAILURE STATE DEFINITION
```

Files 3P2B.RS* give the output from program *harpeng* resulting from successive *harpeng* executions. Each 3P2B.RS* file is the output from a subsequent execution of *harpeng* with different multifault model specifications.

::::::::::::::

3P2B.RS1

::::::::::::::

This listing is for 3P2B.RS1 for the same system previously presented with no near-coincident fault model invoked.

```
                ----------------- HARP ----------------------
                - The Hybrid Automated Reliability Predictor -
                ------------ Release Version 7.0 -------------
                --------------- February 1993 ----------------

 Modelname:
     3P2B
  Input description (from dictionary file):
  Component type: 1       Name: PROCESSOR
  Symbolic failure rate:
  LAMBDA        Constant failure rate:
                      0.50000000D-02      +/-       0.00000000D+00
  FEHM file name: ESPN.FHM
      For this FEHM model, the exit probabilities are:
         (in the absence of near-coincident faults)
           Transient restoration:        0.50000000D+00   *These values could*
           Permanent coverage:           0.36375000D-01   *change with each  *
           Single-point failure:         0.46362500D+00   * subsequent run.  *
                                                          * (see section 10) *
  Component type: 2       Name: BUS
  Symbolic failure rate:
  MU            Constant failure rate:
                      0.50000000D-01      +/-       0.00000000D+00
  FEHM file name: VALUES
  Symbolic values:
  C2            Coverage factor, value directly specified:
                      0.50000000D+00      +/-       0.00000000D+00
  R2            Restoration factor, value directly specified:
                      0.30000000D+00      +/-       0.00000000D+00
```

```
N2          NCF factor, value directly specified:
                  0.00000000D+00     +/-       0.00000000D+00
S2          SPF factor, value directly specified:
                  0.20000000D+00     +/-       0.00000000D+00
NO near-coincident faults considered.
GERK report E201, Tolerances reset:    0.100D-08   0.100D-08
Time(in Hours):    0.100D+02
State Probabilities
State name: F1            0.13159858D-06       *These values could change*
State name: F2            0.81088630D-01       *(see section 10)          *
State name: FSPF          0.19924322D+00
State name: FNCF          0.00000000D+00
            ------------------------------------------------
              Reliability =   0.71966802D+00
            Unreliability =   0.28033198D+00
Total failure by redundancy exhaustion =   0.81088762D-01
GERK ODE solver:  global error value   0.734D-11
                 relative error value   0.100D-08
See Users Guide, section 3.3 for interpretation.
       1 Reports from the GERK ODE solver.
::::::::::::::
3P2B.RS2
::::::::::::::
```

This listing is for the 3P2B.RS2 file for the same previously described system with the ALL-INCLUSIVE multifault model invoked. The state probabilities, reliability, and unreliability values will change in each subsequent run. (See chapter 10.)

```
                ----------------- HARP -----------------------
                - The Hybrid Automated Reliability Predictor -
                ------------ Release Version 7.0 -------------
                --------------- February 1993 ----------------
Modelname:
   3P2B
Input description (from dictionary file):
Component type: 1      Name: PROCESSOR
Symbolic failure rate:
LAMBDA       Constant failure rate:
```

```
                    0.50000000D-02      +/-       0.00000000D+00
FEHM file name: ESPN.FHM
    For this FEHM model, the exit probabilities are:
       (in the absence of near-coincident faults)
         Transient restoration:        0.50000000D+00  *These could change*
         Permanent coverage:           0.36375000D-01  *(see section 10)  *
         Single-point failure:         0.46362500D+00
Component type: 2      Name: BUS
Symbolic failure rate:
MU          Constant failure rate:
                    0.50000000D-01      +/-       0.00000000D+00
FEHM file name: VALUES
Symbolic values:
C2          Coverage factor, value directly specified:
                    0.50000000D+00      +/-       0.00000000D+00
R2          Restoration factor, value directly specified:
                    0.30000000D+00      +/-       0.00000000D+00
N2          NCF factor, value directly specified:
                    0.00000000D+00      +/-       0.00000000D+00
S2          SPF factor, value directly specified:
                    0.20000000D+00      +/-       0.00000000D+00
ALL-INCLUSIVE near-coincident fault rate used.
Time(in Hours):    0.100D+02
State Probabilities
State name: F1              0.13157227D-06        *These values could change*
State name: F2              0.81088606D-01        *(see section 10)  *
State name: FSPF            0.19923770D+00
State name: FNCF            0.59530150D-05
             ------------------------------------------------
               Reliability =   0.71966761D+00
             Unreliability =   0.28033239D+00
Total failure by redundancy exhaustion =   0.81088738D-01
GERK ODE solver:  global error value   0.348D-13
                 relative error value   0.100D-08
See Users Guide, section 3.3 for interpretation.
```

```
        0 Reports from the GERK ODE solver.

::::::::::::::

3P2B.RS3

::::::::::::::
```

This listing is for the 3P2B.RS3 file for the previous system with SAME-type multifault model invoked.

```
                ----------------- HARP ----------------------
                - The Hybrid Automated Reliability Predictor -
                ------------ Release Version 7.0 -------------
                --------------- February 1993 ----------------
 Modelname:
    3P2B
  Input description (from dictionary file):
  Component type: 1      Name: PROCESSOR
  Symbolic failure rate:
  LAMBDA       Constant failure rate:
                      0.50000000D-02     +/-       0.00000000D+00
  FEHM file name: ESPN.FHM
      For this FEHM model, the exit probabilities are:
         (in the absence of near-coincident faults)
           Transient restoration:       0.50000000D+00
           Permanent coverage:          0.36375000D-01
           Single-point failure:        0.46362500D+00
  Component type: 2      Name: BUS
  Symbolic failure rate:
  MU           Constant failure rate:
                      0.50000000D-01     +/-       0.00000000D+00
  FEHM file name: VALUES
  Symbolic values:
  C2           Coverage factor, value directly specified:
                      0.50000000D+00     +/-       0.00000000D+00
  R2           Restoration factor, value directly specified:
                      0.30000000D+00     +/-       0.00000000D+00
  N2           NCF factor, value directly specified:
                      0.00000000D+00     +/-       0.00000000D+00
  S2           SPF factor, value directly specified:
```

```
                    0.20000000D+00      +/-       0.00000000D+00
 SAME-TYPE near-coincident fault rate used.
 Time(in Hours):    0.100D+02
 State Probabilities
 State name: F1             0.13159661D-06
 State name: F2             0.81088627D-01
 State name: FSPF           0.19924267D+00
 State name: FNCF           0.59152289D-06
             ------------------------------------------------
               Reliability =   0.71966798D+00
             Unreliability =   0.28033202D+00
 Total failure by redundancy exhaustion =   0.81088759D-01
 GERK ODE solver:  global error value   0.348D-13
                  relative error value   0.100D-08
 See Users Guide, section 3.3 for interpretation.
        0 Reports from the GERK ODE solver.
::::::::::::::
3P2B.RS4
::::::::::::::
```

This listing is for the 3P2B.RS4 file for the same previous system with the USER-defined multifault model invoked.

```
                ----------------- HARP ----------------------
                - The Hybrid Automated Reliability Predictor -
                ------------ Release Version 7.0 -------------
                --------------- February 1993 ----------------
 Modelname:
    3P2B
 Input description (from dictionary file):
 Component type: 1      Name: PROCESSOR
 Symbolic failure rate:
 LAMBDA       Constant failure rate:
                    0.50000000D-02      +/-       0.00000000D+00
 FEHM file name: ESPN.FHM
     For this FEHM model, the exit probabilities are:
        (in the absence of near-coincident faults)
          Transient restoration:       0.50000000D+00
```

```
          Permanent coverage:          0.36375000D-01
          Single-point failure:        0.46362500D+00
     INTERFERING COMPONENT TYPES: 2
Component type: 2       Name: BUS
Symbolic failure rate:
MU          Constant failure rate:
                  0.50000000D-01      +/-       0.00000000D+00
FEHM file name: VALUES
     INTERFERING COMPONENT TYPES:
Symbolic values:
C2          Coverage factor, value directly specified:
                  0.50000000D+00      +/-       0.00000000D+00
R2          Restoration factor, value directly specified:
                  0.30000000D+00      +/-       0.00000000D+00
N2          NCF factor, value directly specified:
                  0.00000000D+00      +/-       0.00000000D+00
S2          SPF factor, value directly specified:
                  0.20000000D+00      +/-       0.00000000D+00
INTERFERING COMPONENT TYPE near-coincident rate.
Time(in Hours):    0.100D+02
State Probabilities
State name: F1              0.13157423D-06
State name: F2              0.81088609D-01
State name: FSPF            0.19923825D+00
State name: FNCF            0.53616071D-05
            ------------------------------------------------
              Reliability =   0.71966765D+00
            Unreliability =   0.28033235D+00
Total failure by redundancy exhaustion =   0.81088740D-01
GERK ODE solver:  global error value   0.348D-13
                  relative error value   0.100D-08
See Users Guide, section 3.3 for interpretation.
       0 Reports from the GERK ODE solver.
```

FEHM file ESPN.FHM after running *harpeng*.

```
HARP.SINGLE.FAULT.MODEL
COVERAGE INPUT PARAMETERS:
TIME                              DISTRIBUTION AND PARAMETERS
----                              ---------------------------
ACTIVE TRANSITION                 UNIF      0.      1.000
BENIGN TRANSITION                 UNIF      0.      .5000
TRANSIENT LIFETIME                EXP     100.0       0.
DETECT TRANSITION                 UNIF      0.      .4000
ERROR TRANSITION                  WEBUL   10.00     2.500
ERROR-DETECT TRANSITION           WEBUL   50.00     .2500
ISOLATION TRANSITION              NORML   4.000     1.000
RECOVERY TRANSITION               ERLNG   100.0     2.000
RECONFIGURATION TRANSITION        NORML   1.000     .5000
OTHER PARAMETERS:
PROBABILITY OF FAULT DETECTION BY SELF TEST: 0.9000
PROBABILITY OF ERROR DETECTION: 0.9000
PROB. OF ISOLATING DETECTED FAULT: 0.9000
NUMBER OF RECOVERY ATTEMPTS:      5
PROB. OF SUCCESSFUL RECONFIGURATION: 0.9000
FRACTION OF FAULTS WHICH ARE TRANSIENT: 0.5000
FRACTION OF FAULTS WHICH ARE PERMANENT: 0.4000
DESIRED CONFIDENCE LEVEL: 90%
ALLOWABLE ERROR: 10%
** Cut here if parameters in FEHM have changed      **
** to obtain new simulation results. Rerun harpeng. **
SIMULATION RESULTS:
R EXIT:
PROB[REACHING EXIT]: LOW=0.49080417      NOM=0.50000000      HIGH=0.50919583
1ST MOMENT:          LOW=0.48844865E-02 NOM=0.51583926E-02 HIGH=0.54322987E-02
2ND MOMENT:          LOW=          0.   NOM=0.13748132E-03 HIGH=0.15771914E-02
3RD MOMENT:          LOW=0.52741874E-05 NOM=0.52741874E-05 HIGH=0.52741874E-05
C EXIT:
PROB[REACHING EXIT]: LOW=0.32931688E-01 NOM=0.36375000E-01 HIGH=0.39818312E-01
1ST MOMENT:          LOW= 3.2281735      NOM= 3.5878440      HIGH= 3.9475146
```

2ND MOMENT:	LOW= 0.	NOM= 26.736224	HIGH= 119.16996
3RD MOMENT:	LOW= 0.	NOM= 202.06828	HIGH= 1027.6762
S EXIT:			
PROB[REACHING EXIT]:	LOW=0.45098586	NOM=0.46362500	HIGH=0.47626414
1ST MOMENT:	LOW= 3.2217696	NOM= 3.2808923	HIGH= 3.3400150
2ND MOMENT:	LOW= 0.	NOM= 25.452089	HIGH= 186.29844
3RD MOMENT:	LOW= 0.	NOM= 0.	HIGH= 0.

Appendix B

Modeling Advanced Fault-Tolerant Systems With HARP

Since the original draft of the tutorial was written, the HARP developers have explored the possible uses of the dynamic fault tree gates. Many of models involving the use of the dependency fault tree gates were published in several conference proceedings and journals. One of these papers (ref. 23) is included in this appendix to illustrate the powerful modeling flexibility of the dynamic fault tree gates (ref. 24) and to encourage the reader to further explore their applications through the published literature (refs. 25 to 30). Part of the work embodied in this paper is the work of the U.S. Government and thus may be used for government purposes; any other use is not authorized.

1991 *Annual* RELIABILITY AND MAINTAINABILITY *Symposium*

Modeling Advanced Fault-Tolerant Systems with HARP

Joanne Bechta Dugan

S. J. Bavuso & M. A. Boyd

Dr. Joanne Bechta Dugan
Dept. of Computer Science
Duke University
Durham, North Carolina 27706 USA

FTS

Modeling Advanced Fault Tolerant Systems with HARP

Summary

Reliability analysis of fault tolerant computer systems for critical applications is complicated by several factors. In this tutorial, we discuss these modeling difficulties and describe and demonstrate dynamic fault tree modeling techniques for handling them. The techniques described include behavioral decomposition, a Markov solution of a fault tree, and the use of special purpose gates in the fault tree to model sequence dependent behavior. Several advanced fault tolerant computer systems are described, and fault tree models for their analysis are presented. These systems include a loosely-coupled distributed system, a system of fault tolerant building blocks, a fault tolerant parallel processor, a mission avionics system and several instances of fault tolerant hypercube architectures. HARP (the Hybrid Automated Reliability Predictor) is a software package developed at Duke University and NASA Langley Research Center that is capable of solving the fault tree models presented int this tutorial.

Knowledge of fault tree and Markov modeling is assumed. The emphasis of this tutorial is on techniques for constructing a dynamic fault tree model of advanced systems. This fault tree will be solved using Markov methods. We assume that the modeler will use some software package for solution of the model; a mathematical discussion of solution techniques is not included.

Joanne Bechta Dugan

Joanne Bechta Dugan was awarded the B.A. degree in Mathematics and Computer Science from La Salle University, Philadelphia, PA in 1980, and the M.S. and PhD degrees in Electrical Engineering from Duke University, Durham, NC in 1982 and 1984, respectively. An Associate Professor in the Department of Computer Science at Duke University, her research includes modeling and analysis of fault-tolerant computer and control systems. She was a Visiting Scientist at the Center for Digital Systems Research at the Research Triangle Institute, and has been the Principal Investigator of grants from NASA, the Office of Naval Research, the National Science Foundation and Draper Laboratories. Dr. Dugan is an Associate Editor of the IEEE Transactions on Reliability, Senior member of the IEEE, and a member of Eta Kappa Nu and Phi Beta Kappa

Salvatore J. Bavuso

Salvatore J. Bavuso is a senior researcher at NASA Langley Research Center in Hampton, Virginia. He received the BS degree in mathematics from the Florida State university in 1964 and the MS degree in applied mathematics from the North Carolina State University at Raleigh in 1971. He has been instrumental in the development of advanced reliability modeling technology for over a decade and is the NASA project manager for the HARP and CARE III programs.

Mark A. Boyd

Mark A. Boyd was awarded the B.A. degree in Chemistry from Duke University, Durham, NC in 1979, and the M.A. degree in Computer Science from Duke University in 1986. He is currently a Ph.D. candidate in Computer Science at Duke University. His research interests include modeling and analysis of fault-tolerant computer systems and parallel processing. He is a member of the IEEE and the ACM.

Contents

Part I
Background

1 Introduction

Fault tolerant computer systems for critical applications are characterized by several factors which complicate their analysis. Systems designed to achieve high levels of reliability frequently employ high levels of redundancy, dynamic redundancy management, and complex fault and error recovery techniques. In this tutorial we consider advanced fault tree modeling techniques to include these factors in the analysis of system reliability.

In this tutorial, we assume the following

- Faults occur randomly and are statistically independent.
- Lifetime distributions are exponential. Faults occur at a constant average rate, which is referred to as the failure rate of the component.
- Mission lengths are relatively short, so that the probability of more than a few failures is low.
- The systems are not repairable while in use.

Systems which violate these assumptions can be handled by more sophisticated techniques which fall outside the scope of this tutorial.

There are several possible for the reliability analysis of fault tolerant computer systems for critical applications. In addition to predicting the reliability of the system for a specified mission time, these techniques can facilitate tradeoff analysis for various fault tolerant techniques, or can be used to compare alternative architectures for a system still in the design phase. Even if a system exists only as a rough sketch on paper, analysis techniques can be used to analyze parametric sensitivity in order to determine which factors have the strongest impact on the reliability of the system.

Fault trees are frequently used for reliability analysis of critical systems. Fault tree models are well accepted and solution methods are well known, but exact analysis of fault trees with many basic events is often expensive, both in terms of developing the model and in solving the model once it is developed. Also, several important types of dynamic behavior in advanced fault tolerant systems cannot be adequately captured in a standard fault tree model. These dynamic behaviors include transient recovery, intermittent errors, and sequence dependency. Markov models present an alternative modeling technique that is flexible enough to model nearly any such dynamic system. Tools and techniques exist for the solution of even very large Markov models. However, the construction of a Markov model for any but the simplest system can be tedious and error prone.

To exploit the relative advantages of both fault trees and Markov models, while avoiding many of the shortcomings, we define a model that is flexible enough to capture the dynamic aspects of the system, but which is (almost) as easy to use as a standard fault tree. The model construction and solution process is facilitated by the new model in three major ways, which are defined and demonstrated via example in this tutorial.

- ***Behavioral decomposition*** is used to separately define models for system structure and fault recovery.
- Several additional gates are introduced into the fault tree model to capture dynamic behavior.
- The fault tree model of system structure is internally and automatically converted to a Markov model, to which is added the fault recovery information.

These techniques have been implemented in HARP (the Hybrid Automated Reliability Predictor), a software package for the analysis of advanced fault tolerant systems, developed by NASA Langley Research Center and Duke University.

The models exampled described are all solved using HARP. For more information about the availability of HARP, contact Sal Bavuso at the NASA Langley Research Center, Mail Stop 478, Hampton, VA, (804) 864-6189. The techniques implemented in HARP are described in more detail in other publications. References [2, 10, 16, 19] are general papers describing HARP. More details of the models presented here, as well as other models using HARP appear in [1, 4, 5, 11, 8]. Modeling the recovery process is covered in detail in [9].

2 Behavioral decomposition

A common approach to modeling complex systems consists of structurally dividing the system into smaller subsystems (e.g. processors, memory units, buses), analyzing the dependability of the subsystems separately, and then combining the subsystem solutions to obtain the system solution. A system level analysis can then be effected by analyzing each subsystem separately and combining the results to obtain the final solution. This structural decomposition is allowed only if the subsystems' fault tolerant behaviors are mutually statistically independent.

An alternative to such a structural decomposition is ***behavioral decomposition***. Generally, the time scale for the occurrence of faults and their associated errors is relatively long (i.e. weeks or months) while the time scale for recovery is relatively short (milliseconds). Behavioral decomposition exploits this time scale difference, by allowing an analyst to describe the two behavior types (occurrence and recovery) in separate models.

Using behavioral decomposition, the model is decomposed into fault-occurrence and repair (FORM) and fault

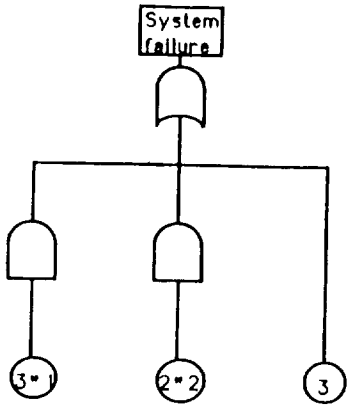


Figure 1: Fault tree model of example system

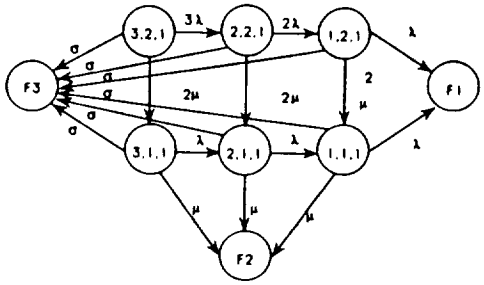


Figure 2: Markov chain model of example system

and error handling (FEHM) submodels. The FORM contains information about the structure of the system and the fault arrival process. The FEHM (often called the *coverage* model) allows for the modeling of permanent, intermittent, and transient faults, and models the on-line recovery procedure necessary for each fault type. We describe this process of model construction by way of a simple three processor, two memory (3P2M) example system.

2.1 The fault occurrence and repair model (FORM)

We wish to model a computer consisting of three processors and two shared memories (3P2M) communicating over a shared bus. The system is operational as long as one processor can communicate with one of the memories. We describe the system structure model as a fault tree, as shown in figure 1, where the top event, System Failure is caused by bus failure OR all processors failing, OR both memories failing. The abbreviation for the combined basic event $i * j$ represents i statistically independent occurrences of component type j.

Figure 2 shows the (continuous time) Markov chain

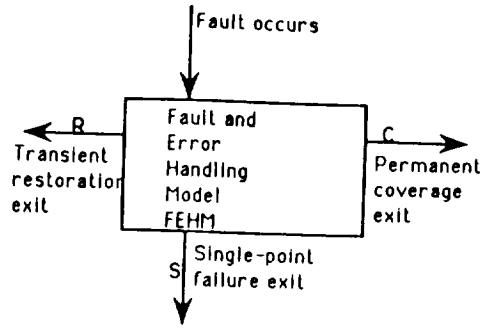


Figure 3: General structure of FEHM

representation of the system whose fault tree is shown in figure 1. The states are labeled with an ordered triple, where element

1. denotes the number of operational processors,
2. denotes number of operational memories, and
3. denotes the state of the bus.

An arc between states (i, j, k) and $(i-1, j, k)$ is labeled with $i * \lambda$ (where λ is the failure rate of processors). Likewise, an arc between states (i, j, k) and $(i, j-1, k)$ is labeled with $j * \mu$ (where μ is the failure rate of memory units). The failure rate of the bus is σ.

F1 represents exhaustion of the processor cluster

F2 represents exhaustion of the memories, and

F3 represents failure of the bus.

The fault tree in figure 1 can be automatically converted to the Markov chain in figure 2. All possible occurrences of basic events that leave the system operational are enumerated; each combination becomes a state in the Markov chain.

The advantage of allowing a fault tree description of the system is that the modeler need not perform the tedious task of determining the Markov chain representation of a system that can be described as a fault tree. Very often, a relatively simple fault tree can give rise to a very large and complicated state space in the corresponding Markov chain. The modeler can use the parsimony of the fault tree representation of the system to generate the state space of the Markov chain automatically, and then make adjustments to the Markov chain as needed.

2.2 The fault and error handling model (FEHM)

We next concentrate on modeling the detailed behavior of the system when a fault occurs. The general structure

of a model that represents the recovery process that is initiated when a fault occurs is shown in figure 3. The entry point to the model signifies the occurrence of the fault, and the three exits signify three possible outcomes. The transient restoration exit (labeled R) represents the correct recognition of and recovery from a transient fault. A transient is usually caused by external or environmental factors, such as excessive heat or a "glitch" in the power line. It is generally believed that the vast majority of faults are transient. Successful recovery from a transient fault restores the system to a consistent state without discarding any components, for example by retrying an instruction or rolling back to a previous checkpoint. Reaching this exit successfully requires timely detection of an error produced by the fault, performance of an effective recovery procedure, and the swift disappearance of the fault (the cause of the error).

The permanent coverage exit (labeled C) denotes the determination of the permanent nature of the fault, and the successful isolation and removal of the faulty component. The single point failure exit (labeled S) is reached when a single fault causes the system to crash. This generally occurs if an undetected error propagates through the system, or if the faulty unit cannot be isolated and thus the system cannot be reconfigured.

As an example of a FEHM for the memory subsystem of figure 1, assume that single-bit memory errors (which are 98% of all memory faults) can be masked and faults that affect more than one memory bit are 95% detectable. Upon detection of a multiple memory error, the affected portion of memory is discarded, the memory mapping function is updated, and the needed information is reloaded from a previous checkpoint and updated to represent the current state of the system. The first two moments [1] of the time to perform this recovery have been determined by experiment to be 0.45 and 0.25 (time scale in seconds). Experimentation also revealed that this recovery from the detected multiple memory errors is 85% effective. It follows that a memory fault causes a single point failure (in zero time) with probability 0.00385 [= 0.02 * (0.05 + 0.95*0.15)] if it causes multiple errors and is not detected or is not recoverable. This behavior can be captured in a FEHM model by providing the probability of reaching each of the three exits and by providing the first few moments of the time to reach each exit. Figure 4 is a pictorial representation of the recovery process for the memory subsystem.

The recovery process for the faults that occur in the processor is more complex. When a fault occurs in a processor, a multi-step recovery process commences.

1. Wait for 0.1 second and do nothing, in the hope that the fault is transient and will disappear.

[1] If, in successive experiments, recovery times are $T_1, T_2, \ldots, T_k$, then the mean (first moment) is $\frac{1}{k}\sum_{j=1}^{k} T_j$ and the second moment is $\frac{1}{k}\sum_{j=1}^{k} T_j^2$.

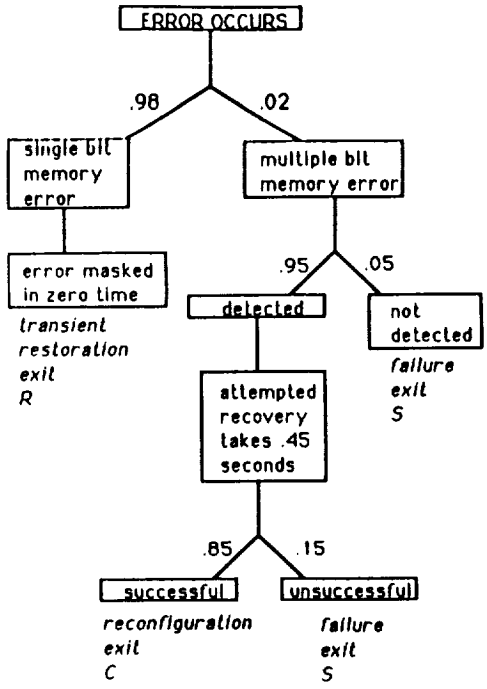


Figure 4: Recovery model for memory subsystem

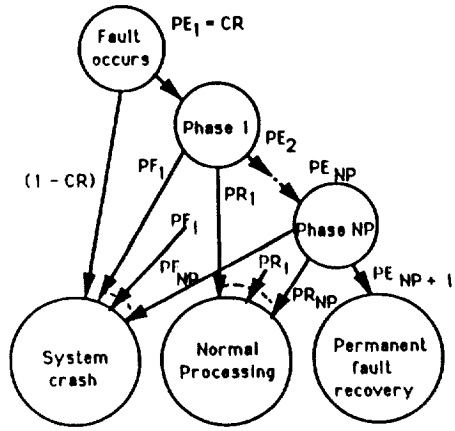


Figure 5: ARIES transient fault recovery model

2. Retry the offending computation several times, which takes, say 0.5 seconds.

3. If the fault still persists, a rollback to a previous checkpoint followed by recomputation is performed, taking 2 seconds total.

4. If the fault still persists then it is assumed to be permanent, and a permanent recovery process is initiated.

In all, there are up to 3 phases of transient recovery, possibly followed by a permanent recovery. The 3 phases of recovery succeed (i.e. the system is operational at the conclusion of the phase) with probabilities 0, 0.5, and 0.8, respectively; the permanent recovery process succeeds with probability 0.875. The average lifetime of a transient fault is 0.5 seconds. The ARIES transient recovery model (shown in figure 5) represents a multi-phase recovery process that executes n successive recovery phases, and is used to model the recovery process for processor failures. The parameters shown on the ARIES model in figure 5 are calculated from the input parameters that describe the recovery process.

2.3 Near-coincident faults

In highly reliable systems, such as those used for flight control, the probability of a second fault occurring while attempting recovery from a given fault cannot be ignored. The occurrence of a second, near-coincident fault (while attempting to handle a single fault) causes immediate system failure, if the second and first faults are critically coupled. The modeler must designate which sets of faults are critically coupled, or can assume either extreme: all faults are critically coupled or no faults are critically coupled. Once the set of critically coupled faults has been determined, the calculation of the probability of near-coincident faults is straightforward, given some measure of the time spent in a recovery model. In the 3P2M example, if a processor fails while a memory failure is being handled, or during the recovery from a fault in another processor, the system fails. If, however, a memory fails during a processor recovery, no immediate failure occurs. A bus failure would interfere with processor or memory recovery.

A fourth exit is then added to the FEHM model, representing the occurrence of a near-coincident fault before another exit is reached. Consequently, the probability of reaching one of the original three exits is reduced by a factor equaling the probability that an interfering near-coincident fault does not occur. This single-entry 4-exit model is then automatically inserted into the FORM model, as described in the following section.

Cause of Failure	Probability
Exhaustion of Processors	2.20×10^{-10}
Exhaustion of Memories	1.61×10^{-10}
Exhaustion of Buses	9.99×10^{-6}
Single Point Failure	3.53×10^{-5}
Near-Coincident Faults	4.49×10^{-10}
Total Unreliability	4.53×10^{-5}

Table 1: Solution of 3P2M example system

2.4 Combining FORM and FEHM models

Once the FORM and FEHM models are described, they are then combined. We demonstrate this process for the Markov chain in figure 2 which results from the fault tree in figure 1. For each failure of a redundant component, the appropriate FEHM model is invoked. That is, a FEHM model is inserted on each failure arc between operational states in the Markov chain, as shown in figure 6. In the 3P2M example, the FEHMs on the horizontal failure arcs are copies of the ARIES model (figure 5), while the FEHMs on the vertical failure arcs are copies of the memory coverage model (figure 4). Two failure states are inserted:

- *FSPF* denoting the occurrence of a single-point failure, and
- *FNCF* denoting the occurrence of critically coupled near-coincident faults.

Each FEHM model is then solved for the probability of reaching each of its three exits, and the FEHM model is replaced by a branch point. The resulting Markov chain (see figure 7) is then solved for the reliability of the system, which is given by the probability that the system is not in any failure state.

Table 1 shows the results of the reliability analysis for a 10 hour mission of the 3P2M example. For this model, we assume that the failure rate of the processor $\lambda = 10^{-4}$, for the memory $\mu = 10^{-5}$ and for the bus $\sigma = 10^{-6}$. The largest contributor to the unreliability is single-point failure, that is, faults from which recovery is not successful.

3 Dynamic fault-tree gates

A major disadvantage of traditional fault tree analysis is the inability of standard fault tree models to capture sequence dependencies in the system, and still allow an analytic solution. As an example of a sequence dependent failure, consider a system with one active component and one standby spare connected with a switch controller [15]. If the switch controller fails after the active unit fails (and thus the standby is already in use), then the system can continue operation. However, if the switch controller fails before the active unit fails, then the standby unit cannot be switched into active operation and the system fails.

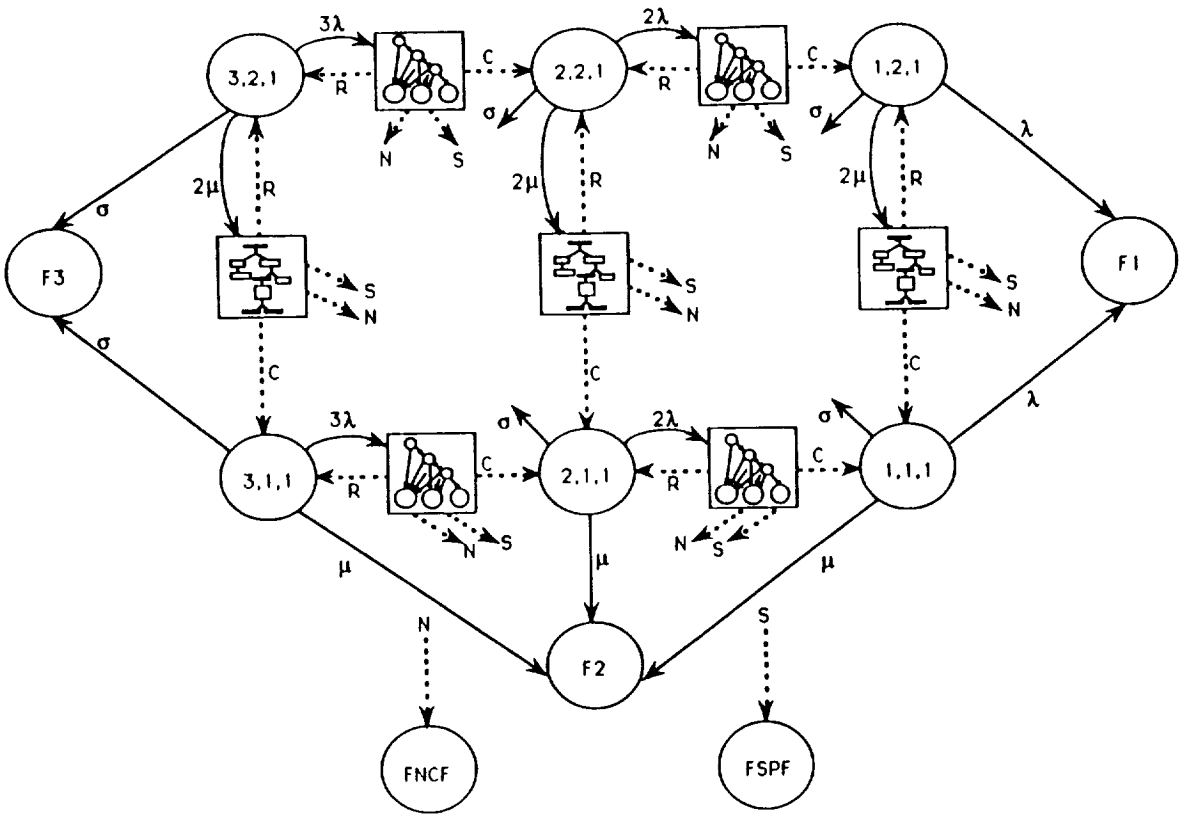


Figure 6: Combination of FEHM and FORM models

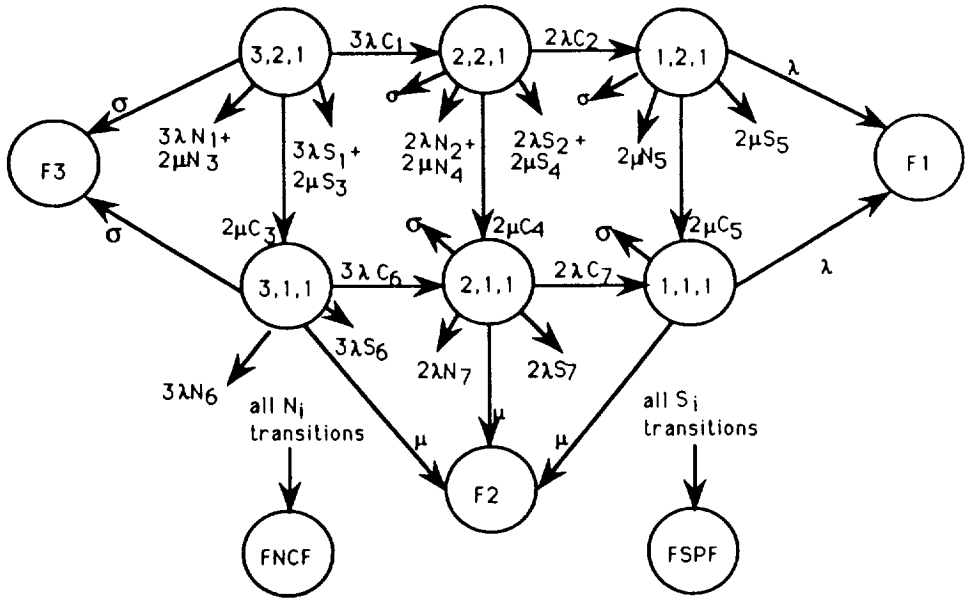


Figure 7: Reduction of combined FEHM and FORM models

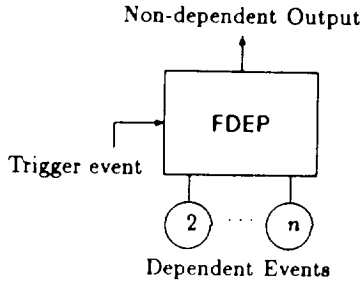


Figure 8: Functional dependency gate

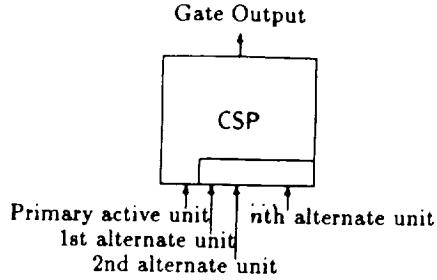


Figure 9: Cold spare gate

Thus, the failure criteria depend not only on the combinations of events, but also on the sequence in which events occur.

Systems with various sequence dependencies are usually modeled with Markov models. If, instead of using standard fault tree solution methods, the fault tree is converted to a Markov chain for solution, the expressive power of a fault tree can be expanded by allowing certain kinds of sequence dependencies to be modeled by defining special purpose gates to capture specific types of sequence dependent behaviors. There are several different kinds of sequence dependencies in fault tolerant systems. This section identifies several such dependencies, and defines specific gates to express these behaviors in fault tree models. Part II demonstrate the use of these gate types in several examples.

3.1 Functional dependency gate

Suppose that a system is configured such that the occurrence of some event (call it a *trigger* event) causes other dependent components to become inaccessible or unusable. In this case, later failures of the dependent components will not further affect the system and should not be considered. A *functional dependency* gate (see figure 8) has a single trigger input (either a basic event or the output of another gate in the tree), a non-dependent output (reflecting the status of the trigger event) and one or more dependent basic events. The dependent basic events are functionally dependent on the trigger event. When the trigger event occurs, the dependent basic events are forced to occur. In the Markov chain generation, when a state is generated in which the trigger event is satisfied, all the associated dependent events are marked as having occurred. The occurrence of any of the dependent basic events has no effect on the trigger event.

The functional dependency gate is useful where communication is achieved through some network interface elements, where the failure of the network element isolates the connected components. In this case, the failure of the network element is the trigger event and the connected components are the dependent events. Part II describes several applications of the functional dependency gate.

3.2 Cold spare gate

Consider a system that utilizes cold spares, that is, spare components that are unpowered, and thus do not fail before being used. Such systems cannot be modeled exactly using standard fault tree techniques because the system failure criteria cannot be expressed in terms of combinations of basic events, all using the same time frame.

We address this fault tree deficiency by introducing a *cold spare* gate (see figure 9), with one primary input and one or more alternate inputs. All inputs are basic events. The primary input is the one that is originally powered on, and the alternate input(s) specify the (initially unpowered) components that are used as replacements for the primary unit. The cold spare gate has one output which becomes true after all the input events occur.

The conversion of the fault tree to a Markov chain makes the consideration of cold spares possible. In a state where the primary unit is operational, the cold spares are not permitted to fail. However, once the primary unit has failed, then the first alternate unit can fail. After the first alternate fails, the remaining alternates are allowed to fail, one at a time in the order specified, until the spares are exhausted. The possibility of being unable to reconfigure correctly the spare unit into operation is captured in the (separately specified) coverage model.

The functional dependency gate and the cold spare gate can interact in an interesting way. Suppose that the spare units are functionally dependent on some other (otherwise unrelated) component. The occurrence of the trigger event can render one or more of the spares unusable, even if they have not been switched into active operation yet. Then, if the primary unit fails, the spares are unavailable to replace it. This is the one case where a spare can "fail" even while it is unpowered. Part II gives examples of the use of the cold spare gate.

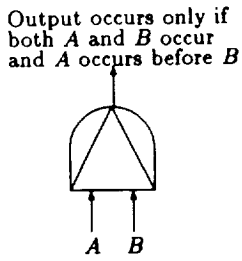


Figure 10: Priority-AND gate

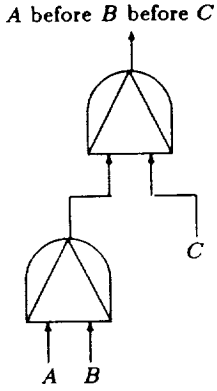


Figure 11: Cascading priority-AND gates

3.3 Priority-AND gate

The priority-AND gate is logically equivalent to an AND gate, with the added condition that the events must occur in a specific order. The priority-AND gate (as shown in figure 10) has two inputs, A and B. The output of the gate is true if both A and B have occurred, and if A occurred *before* B. If both events have not occurred, or if B occurred before A then the gate does not fire. To represent the behavior that A occurs before B which occurs before C, the priority-AND gates can be cascaded as shown in figure 11.

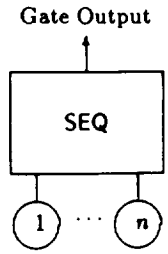


Figure 12: Sequence enforcing gate

3.4 Sequence enforcing gate

The *sequence enforcing gate* forces events to occur in a particular order. The input events are constrained to occur in the left-to-right order in which they appear under the gate (i.e., the leftmost event must occur before the event on its immediate right which must occur before the event on its immediate right is allowed to occur, etc.). There may be any number of inputs (see figure 12), the first of which may be a (possibly replicated) basic event or the output of some other gate. All inputs other than the first are limited to being (possibly replicated) basic events. The sequence enforcing gate can be contrasted with the priority-AND gate in that the priority-AND gate *detects* whether events occur in a particular order (the events can occur in any order) where the sequence enforcing gate will only *allow* the events to occur in a specified order.

In the generation of a Markov chain from a fault tree containing a sequence enforcing gate, states that represent any other ordering than that specified by the sequence enforcing gate are never generated. In part 2 of this tutorial we will show an interesting application of the sequence enforcing gate to model pooled spares.

Part II
Examples

We study several examples of advanced fault tolerant systems, and develop fault tree models to analyze the reliability of these systems. The models are all solved with HARP, the Hybrid Automated Reliability Predictor, developed at NASA Langley Research Center and Duke University. The parameters used for these model and the details of the recovery mechanism are pure conjecture, and should not be interpreted as a factual representation of the parameters associated with the systems.

4 Cm*: a loosely-coupled distributed system

4.1 System description

An instance of the *Cm** system (shown in figure 13) consists of 2 clusters of processors and memories connected by links [18]. Each cluster consists of 4 local switch interface controllers (*S.locals*), each attached to one processor and one 12K memory module. Each processor has 4K of memory on board. The *K.map* is a cluster controller connecting the *S.locals*; the clusters are connected by intercluster communications (*L.inc*). A fault in the *K.map* renders the associated *S.locals* (and their connected processors and memories) inaccessible, while a fault in an *S.local* makes the processor and memory modules connected to it inaccessible.

The *Cm** system exhibits three characteristics that are typical of reliable distributed systems.

1. There are functional interdependencies which can make the development of the fault tree model difficult, for example, the dependence of the accessibility of the processors and memories on the state of the *S.locals*.

2. There are many potential system states: since there are 27 components, the system can be in any one of $2^{27} > 134$ million states, if any component can be in one of two states, functional and failed.

3. There are many failure modes: there are 5405 minimal cut sets for this system (a *cut set* is a set of components whose failure causes the system to fail).

4.2 Failure criteria

The system is considered operational as long as there are 3 processors that can communicate with 3 memories. As long as the *L.inc* is operational, these requirements can be satisfied by the components of both clusters. But, if the *L.inc* fails, the requirements must be met within one cluster.

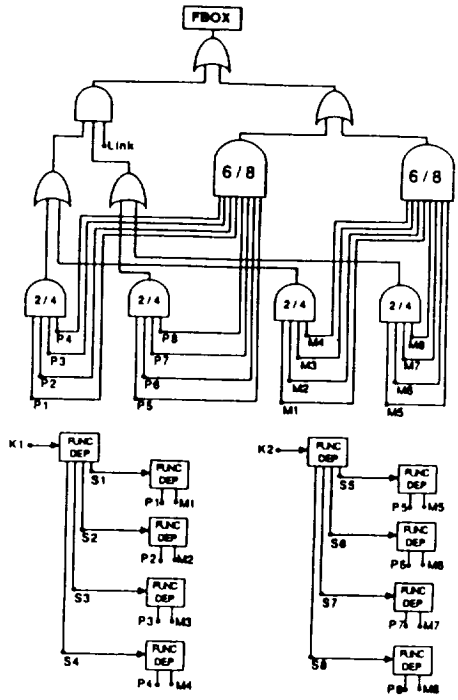


Figure 14: Fault tree model of *Cm** system

4.3 Fault tree model

The development of the fault tree model of the *Cm** system is simplified by the use of a functional dependency gate, to capture the interconnection dependencies. A fault tree model of the Cm* system is shown in figure 14. System failure (the top event) can be attributed to one of two causes which are shown as inputs into the uppermost OR gate. Failure occurs when either the *L.inc* fails and the requirements cannot be satisfied by a single cluster (the left input to the uppermost OR gate), or (independent of the state of the *L.inc*) there are an insufficient total number of processors or memories in both clusters. The output of an m/n gates is true when m of the n input events have occurred.

The functional dependencies of the *S.locals* on the *K.maps* and of the processors and memories on the associated *S.local* are captured in the functional dependency gates (FDEP) shown in figure 14. In this case, there were no explicit reliability requirements concerning the *K.maps* or *S.locals*, so the functional dependency gate is not explicitly connected to the top event in the fault tree. In order to solve a fault tree model containing functional dependency gates via standard combinatorial solution methods, we need to convert the model to a strictly combinatorial one. To accomplish this conversion, the dependency gates can be replaced with OR gates in the following manner. For each occurrence of a dependent basic event, replace that basic event with a logical OR of

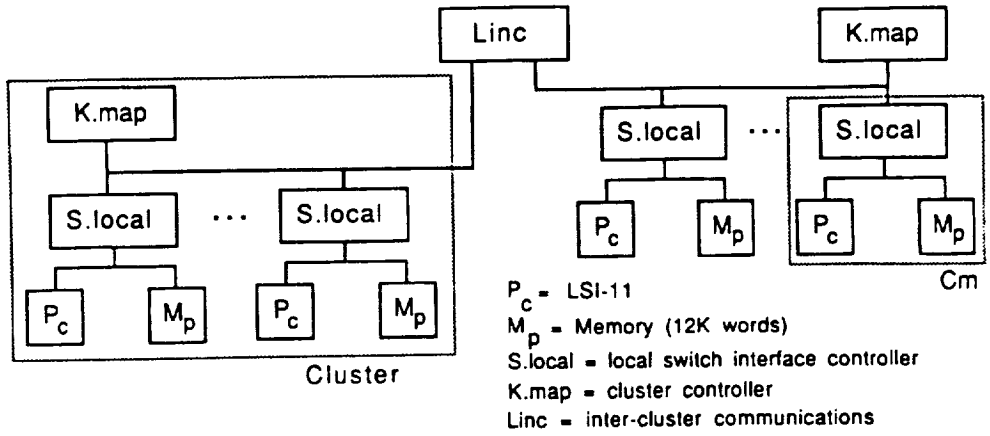


Figure 13: A diagram of the Cm* system.

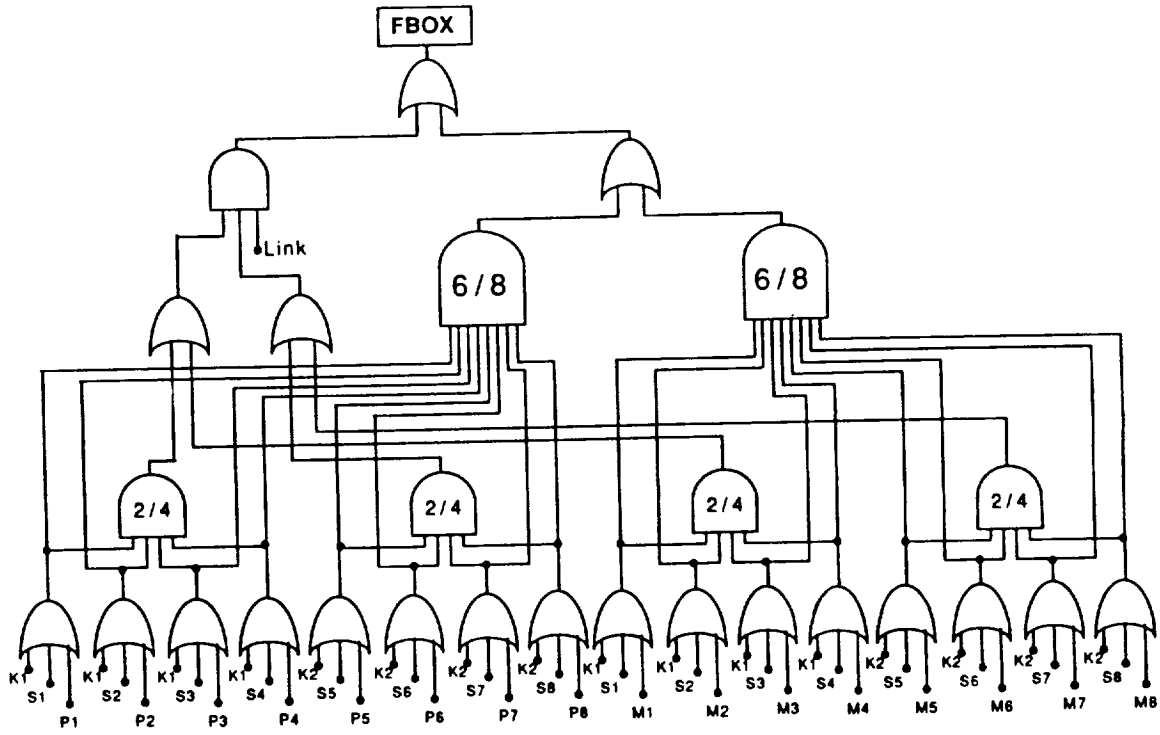


Figure 15: Fault tree model of *Cm** system without functional dependency gates

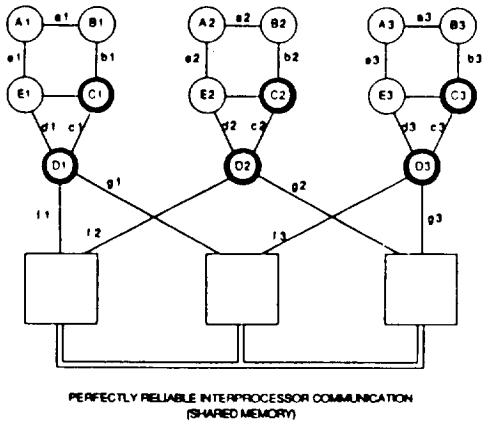


Figure 16: An AIPS I/O network used for example calculations

the basic event and its trigger event. Thus in the *Cm** system, each basic event representing a processor failure is replaced by a logical OR of that processor event, its *S.local* and its *K.map*. Memory events are altered in a similar manner. The fault tree that results from replacing the functional dependency gates is shown in figure 15. The replacement of the functional dependency gates only produces a correct result if no FEHM models are used, that is, if all faults are permanent and are instantaneously and perfectly covered.

5 AIPS: a system of fault-tolerant building blocks

5.1 System description

An example of the AIPS (Advanced Information Processing System) I/O network is shown in figure 16. The AIPS system, designed at the Charles Stark Draper Laboratory, is intended to provide fault-tolerant building blocks that can be used for a variety of real-time control applications [12]. The AIPS I/O network might be used in a flight control system, and consists of 3 rings, each of which contains 5 nodes. Three of the nodes on each ring (those labeled *A*, *B*, *E*) are connected to sensors and/or actuators. Each such device is triplicated, with one copy of each device connected to each ring, via a node in the same location (with the same letter label). The remaining two nodes, *C* and *D*, are termed *root nodes* because they provide the connections to the triplicated computers.

5.2 Failure criteria and parameters

The I/O network fails when

1. Nodes in the same location on two different rings either fail or become isolated from both root connections, OR
2. if 2 of the 3 computers fail or become disconnected from both rings, OR
3. when 2 of the three rings become disconnected from both computers.

As long as a node can communicate with one computer, it can communicate with all computers that are up because the computers are assumed to be connected by a perfectly reliable interconnection mechanism (such as shared memory). For the purpose of this analysis we consider only the I/O network and the computer connections, and not the possible failures of the devices (such as sensors and actuators) connected to the nodes. The failure parameters used for this analysis are

- Node failure rate: 6×10^{-6} per hour
- Link failure rate: 12×10^{-6} per hour
- Computer failure rate: 10^{-4} per hour

5.3 Fault recovery

Recovery from faults in nodes and links is assumed to be perfect and instantaneous. For the computers, however, more detailed coverage modeling is necessary. It is assumed that 85 percent of the faults that occur in the computer system are transient, with the remaining 15 percent being permanent or intermittent in nature. Recovery from computer faults is assumed to be perfect, but not instantaneous: the time to recover from a transient is 1 second, while the time to recover from a permanent or intermittent is uniformly distributed between 1 and 5 seconds. During the recovery interval, if a second, near-coincident fault occurs in either of the other computers, the recovery is interrupted, and system loss is conservatively assumed to occur.

5.4 Fault tree model

The fault tree model of the AIPS I/O network has 102 nodes, including 39 basic events, and is too large to be presented here as a whole. However, figure 17 is a sketch of the fault tree with some of the paths complete. The system fails when one of the seven triplicated subsystems fail (hence seven 2/3 gates are connected to the top OR gate), these being node groups *A* through *E*, the computers, and the root connections between the rings and the computers. A representative of each of the 7 subsystems is shown in detail; the other members of each triplicated subsystem are analogous. The results of the solution of this model appear in table 2.

5.5 Truncated fault tree

An interesting alternative to the development of the full fault tree model is the concept of a *truncated fault tree*.

Full Fault Tree Model Solution AIPS I/O Network Example System		
Truncation Level Size Of Truncated Model	1 Component Failure 42 states, 190 transitions	2 Component Failures 770 states, 5155 transitions
Lower Bound on Unreliability Upper Bound on Unreliability	0.125e-6 2.94e-6	0.126e-6 0.128e-6
Total Run Time	65 CPU seconds	1295 CPU seconds

Table 2: Solution of example AIPS system

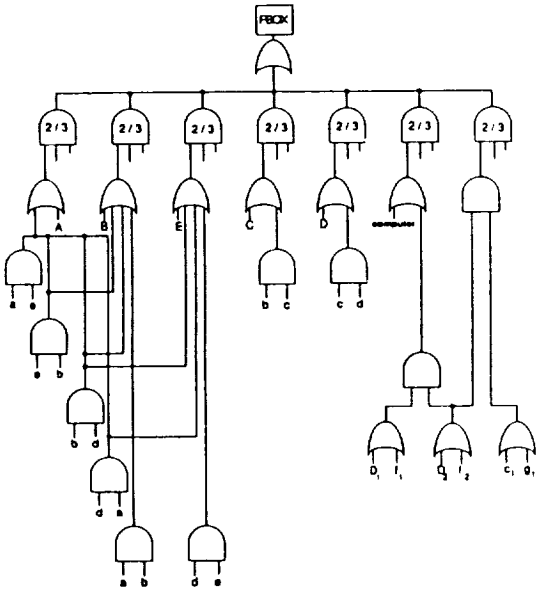


Figure 17: Fault tree model of AIPS I/O network

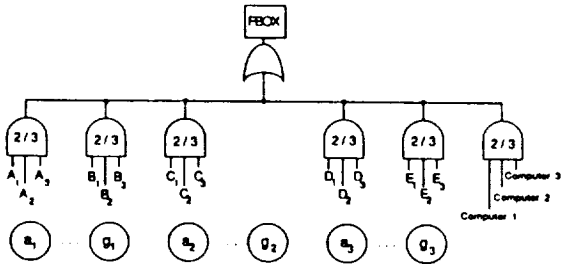


Figure 18: Truncated fault tree model of AIPS network

For the AIPS network (figure 16), the expansion of only 2 failure levels produced a reasonably accurate estimate of the system unreliability. For this case, we could have produced a similar result with a much simpler fault tree, one which explicitly defined only the 2 component failure combinations. Consider the fault tree representation of the AIPS network in figure 18. The top event of this tree is 2-component failure system loss, where the system loss is caused by losing 2 members of any triplicated subsystem. No combination of 2 link failures, or one link failure and one other component failure can lead to system failure, and so the link basic events do not input to any gates in the truncated fault tree. The presence of these *dangling basic events* (basic events that do not input to any gate in the fault tree) can be used to bound the failure probability. If the dangling basic events are ignored then the solution of the fault tree gives an optimistic estimate of the unreliability of the system.

If we are using a strictly combinatorial solution method, we can use the dangling basic events to determine the upper bound on the unreliability by using a k-out-of-n gate. Connect all n basic events (those that are dangling as well as those that are not) to an 3-out-of-n gate (a gate that is activated on the third component failure), and OR its output with the top event of the tree. This is equivalent to assuming that the third component failure causes system failure.

If we need to include the effects of imperfect coverage in the model, we can use the dangling basic events in conjunction with the conversion of the fault tree to a Markov chain. As the Markov chain state space is expanded, all the basic events become part of the state definition. The resulting Markov chain can be used to produce bounds on the unreliability of the system from the solution of the truncated fault tree. It is not necessary in this case to add the m-out-of-n gate as was done with the strictly combinatorial solution. The basic events are simply left dangling. The presence of dangling basic events is crucial to the determination of correct bounds on the system unreliability.

The solution of the truncated Markov chain corresponding to the truncated fault tree of the AIPS system is shown in table 3. A comparison of the numbers in this table with those in table 2, shows that the truncated fault tree can give reasonable results. The time needed by a reliability analyst to determine a truncated fault tree is

Truncated Fault Tree Model Solution AIPS I/O Network Example System		
Truncation Level Size Of Truncated Model	1 Component Failure 42 states, 190 transitions	2 Component Failures 770 states, 4879 transitions
Lower Bound on Unreliability Upper Bound on Unreliability	0.126e-6 0.640e-6	0.1261e-6 0.1263e-6
Total Run Time	58 CPU seconds	1144 CPU seconds

Table 3: Solution of truncated fault tree model of AIPS system

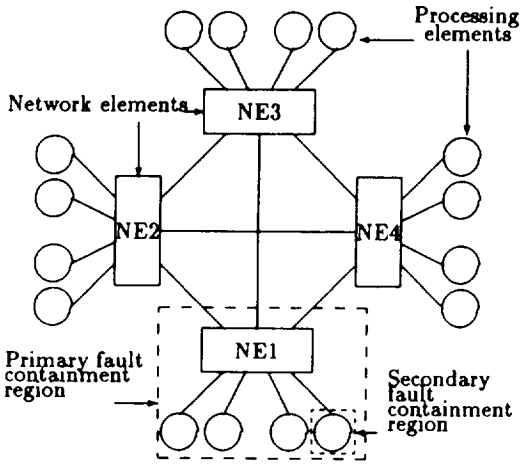


Figure 19: An instance of the fault tolerant parallel processor

substantially less than the time required to derive a complete fault tree model of a system. Further, the combination of a truncated solution technique and a truncated fault tree can allow more faith to be placed in the model, since if there are missing failure combinations they may be accounted for by the bounding technique.

6 FTPP: Fault tolerant parallel processor

6.1 System description

Next we consider several models of the FTPP (Fault Tolerant Parallel Processor) [14, 13] cluster, to compare various configurations of triads with spares. An instance of an FTPP cluster is shown in figure 19, and consists of 16 processing elements (PE), with 4 connected to each of 4 network elements (NE). The network elements are fully connected. In the clusters modeled here, the 16 processors are logically connected to form 4 triads, each with one spare. We investigate three triad/spare configurations, the first two with hot spares and the third with cold spares:

#1 utilizes hot spares; there is one spare for each triad and all spares are attached to the same network element.

 # 2 also uses hot spares; there is one spare on each network element and the spare PE can substitute for any failed PE attached to the same network element.

 # 3 is the same as #1, with all spares on the same NE, but in configuration #3 the spares are cold.

The processing elements in all three configurations functionally depend on the network element to which they are connected. If a network element experiences a permanent failure, the processing elements connected to it are then considered failed.

6.2 Failure criteria and parameters

For all models, a triad fails when it has fewer than 2 active components; the system fails if any triad fails. Failures occur at a constant rate of 1.1×10^{-4} per hour for processing elements, and 1.7×10^{-5} per hour for network elements.

6.3 Fault recovery

Recovery and reconfiguration from faults in processing elements are both perfect, but take a non-zero amount of time. If a second fault occurs in any other component during attempted recovery from a first fault, the system fails. Half of the faults that occur in the processing elements are transient, and can be recovered from without discarding the affected component. The remainder of faults are permanent. The time to recover is exponentially distributed with a mean of 3.6 seconds. Coverage of NE failures is both instantaneous and perfect.

6.4 Fault tree models

6.4.1 Configuration #1

Configuration #1 (shown in figure 20) divides the active elements of a triad among *NE1*, *NE2* and *NE3*, and uses the PE's on *NE4* as spares. The PE's that are in the same relative position on the first three network elements form a triad, and the PE in the same relative position on *NE4* serves as a hot (active) spare for the triad.

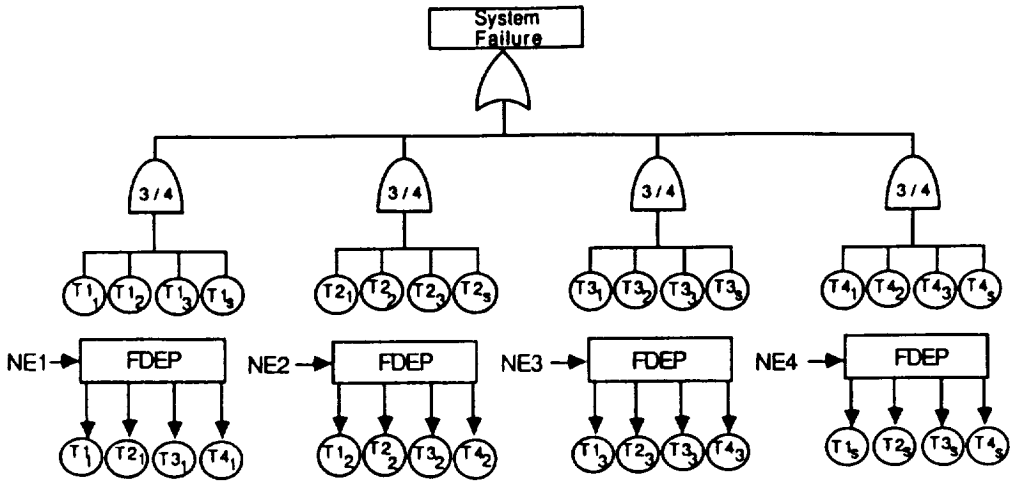


Figure 21: Fault tree model for configuration #1

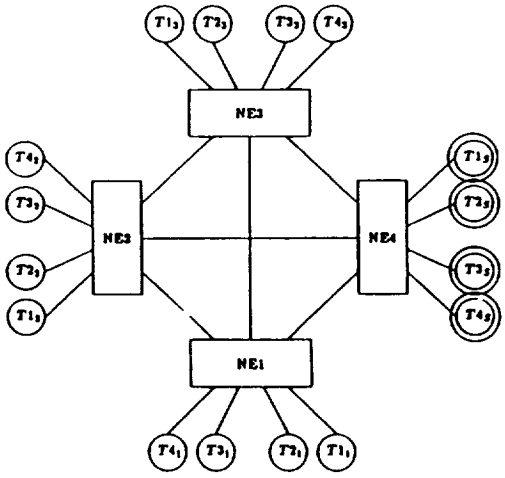


Figure 20: Configuration #1 with one spare per triad

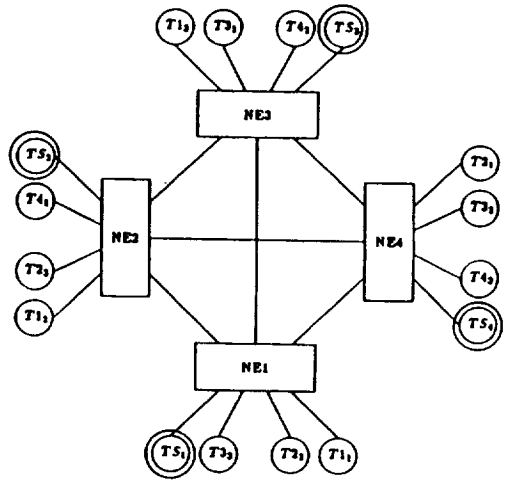


Figure 22: Configuration #2 with one spare per NE

The fault tree model for configuration #1, shown in figure 21, uses four functional dependency gates (FDEP) to reflect the dependence of the processing elements on the network elements. The FDEP gates are not explicitly connected to the other gates in the tree, since the reliability requirements (all 4 triads must be operational) do not explicitly mention the network elements. Figure 21 shows four 3/4 gates connected to the top OR gate, one 3/4 gate for each triad. A triad fails when only one element remains (3 of the 4 elements have failed).

6.4.2 Configuration #2

Configuration #2 is an FTPP cluster with hot spares distributed across the network elements instead of grouped on the same network element (see figure 22). The spare element on each network element can substitute for any failed PE connected to the same NE. That is, processing element TS_1 can substitute for a failed PE connected to *NE1*.

The fault tree model of this system is a bit more complex than the one presented in section 6.4.1, and is shown in figure 23. The functional dependency gates FDEP again reflect the dependence of the processing elements on the network elements. A triad failure is again attributed to losing the majority of operational elements, but it is more difficult to describe the failure of a member of the triad. A member of the triad is failed if it and its spare fail or if its spare is not available when needed. The spare is not available if some other PE on the same NE fails and uses the spare before it is needed by the first PE. For example, in figure 23, the leftmost OR gate that inputs into the leftmost 2/3 gate represents the failure of the first member of the first triad. This member fails if both $T1_1$ (the first member of the first triad) and its spare (TS_1) fail, or if the spare is being used because another failure has already occurred when $T1_1$ fails. The spare will already be in use when $T1_1$ fails if either $T2_2$ or $T3_3$ (the other two active components on the same NE) have failed before $T1_1$ does. This condition is reflected in the Priority-AND gate that inputs to the same OR gate. There is a similar structure of AND and Priority-AND gates to represent the failure of the other members of the triads.

6.4.3 Configuration #3

The third configuration is used to investigate the effect on reliability of keeping the spares unpowered until needed. The FTPP configuration modeled in this section is the same as configuration #1 (figure 20) except that the spares are cold rather than hot. There is one spare for each triad, and all spares are connected to the same network element. The fault tree model for this system, shown in figure 24, uses the cold spare gate. There is one cold spare gate for each member of each triad, where the initially active members of the triad are used as the primary inputs. The basic event representing the cold spare PE is connected to all three cold spare gates since it can substitute for any of the elements.

6.5 Results

This section presents the results obtained from solution of the models of the three FTPP configurations for a mission length of 10 hours. Table 4 contains a compares the reliability of the three configurations. We solved a truncated model (described in more detail later in this section) which produces bounds on the unreliability from a partial solution of the model. Table 4, shows the bounds on the unreliability, and the best case (optimistic) estimate of the probabilities of exhaustion of network elements (exh NE), exhaustion of processing elements (exh PE) and near coincident failures (NCF).

Configuration #2 (that distributed the hot spares across the network elements) not only required a more complicated fault tree for analysis, but also was appreciably less reliable than configuration #1. In configuration #2, the failure of 2 network elements (alone) can kill the system, since the failure of 2 network elements removes 2 members from at least one triad. For example, if *NE1* and *NE2* both fail, then $T1_1$ and $T1_2$ are both disabled, and no spare is available to replace them (because of the functional dependencies). The solution of the model for configuration #2 shows that the predominant cause of failure is the exhaustion of network elements. In configuration #1, the loss of 2 network elements (alone) does not cause any triad to fail, even though it can render all the spare elements unusable.

In the #3 configuration, the spare elements remained unpowered until needed, resulting in a modest decrease in unreliability. Since near-coincident failures contributed more highly to the unreliability of the system, the effect of keeping the PEs unpowered was not as significant as might be expected.

For all three models, the Markov chain was truncated after the consideration of 2 or 3 faults, and so a pair of bounds on the actual reliability were generated. The bounds were tight enough after only considering 2 faults for configuration #2, but we needed to consider a larger model for the other two cases. The reason that the bounds were tighter for configuration #2 is that there were a significant number of failure states encountered when only considering 2 component failures. In the #1 and #3 configurations, there were not many failure states with only 2 failed components. Unfortunately, the number of states in a Markov chain increases exponentially with the number of component failures considered, so the increase in accuracy is accompanied by a large increase in solution times. Table 5 compares the results obtained from the smaller model (truncated after 2 failures) and the larger model (truncated after 3 failures), as well as the size of the models and the run time for the complete generation and solution of the model on a DECstation 3100.

7 ASID MAS: a mission avionics system

7.1 System Description

The ASID (Advanced System Integration Demonstration) project was the first large scale effort in the development of the PAVE PILLAR architecture for advanced tactical fighters. The Boeing Military Airplane Company was one of five contractors who designed implementations of the

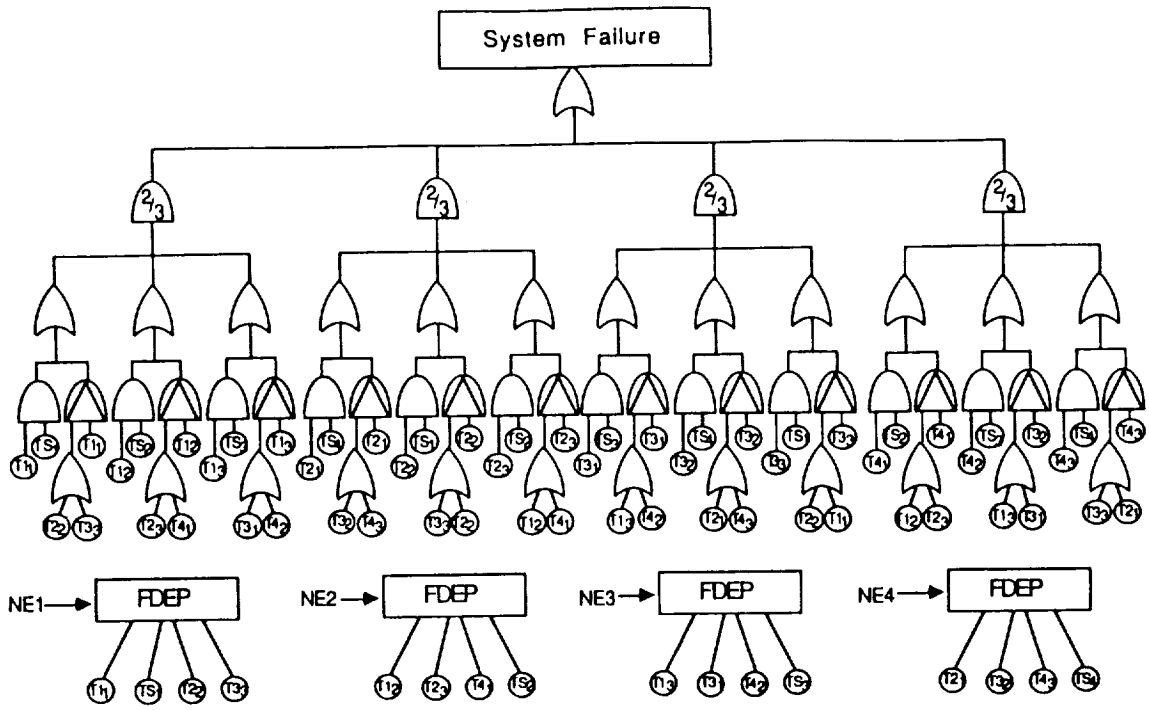


Figure 23: Fault tree model for configuration #2

Configuration	#2: Hot spare per NE	#1: Hot spare per triad	#3: Cold spare per triad
(Best Case) Unreliability	0.207×10^{-6}	0.406×10^{-7}	0.264×10^{-7}
(Worst Case) Unreliability	0.417×10^{-6}	0.407×10^{-7}	0.266×10^{-7}
(Best case) exh. NE	0.174×10^{-6}	0.135×10^{-8}	0.104×10^{-8}
(Best case) exh. PE	0.327×10^{-8}	0.910×10^{-8}	0.705×10^{-8}
(Best case) NCF	0.302×10^{-7}	0.302×10^{-7}	0.183×10^{-7}

Table 4: Results of the solution of all three FTPP models

Configuration	#2: Hot spare/NE	#1: Hot spare/triad	#3: Cold spare/triad
Truncated at 2 component failures			
(Best Case) Unreliability	0.207×10^{-6}	0.406×10^{-7}	0.263×10^{-7}
(Worst Case) Unreliability	0.417×10^{-6}	0.242×10^{-6}	0.132×10^{-6}
Number of states	201	123	225
Number of transitions	877	581	817
Runtime (CPU seconds)	138	99	99
Truncated at 3 component failures			
(Best Case) Unreliability	*analysis not necessary for this example*	0.406×10^{-7}	0.264×10^{-7}
(Worst Case) Unreliability		0.407×10^{-7}	0.266×10^{-7}
Number of states		961	2307
Number of transitions		5469	9777
Runtime (CPU seconds)		2653	5055

Table 5: Comparison of accuracy and model size

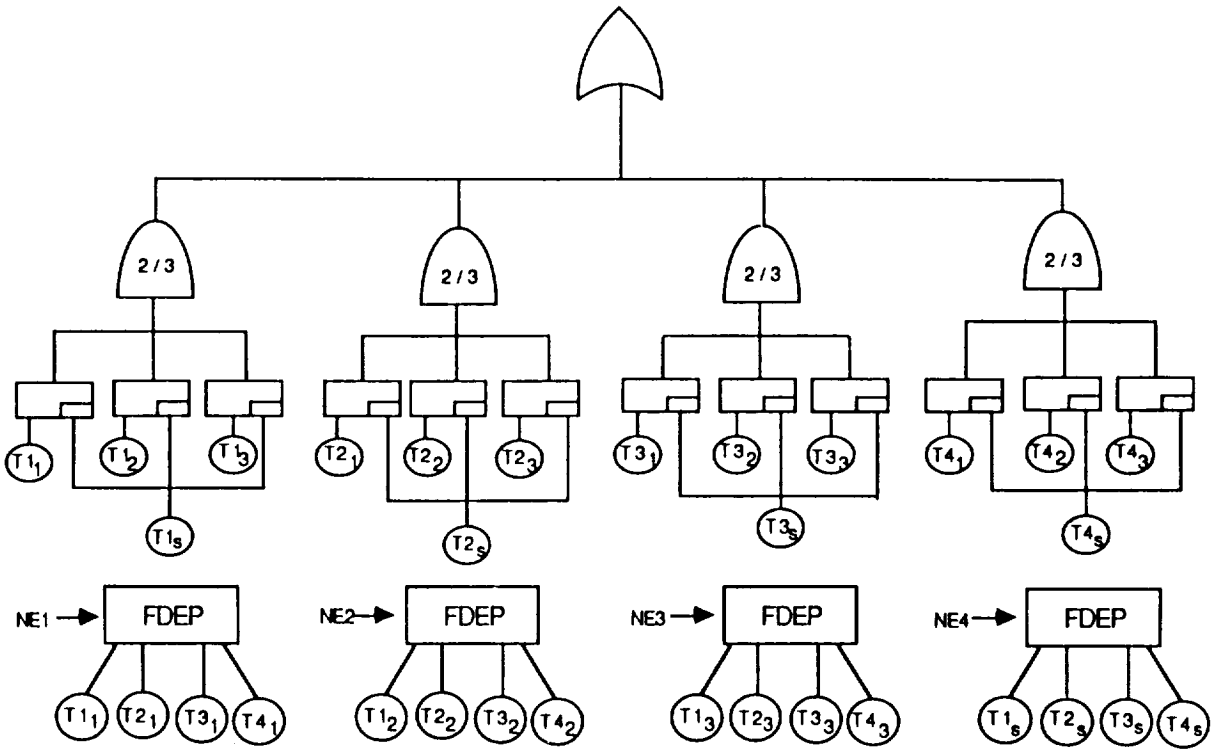


Figure 24: Fault tree model for configuration #3 with one COLD spare per triad

PAVE PILLAR project. A unique feature of the Boeing implementation [3] is the use of dual processor pairs wherever a single processor is required. This processor-pair uses comparison monitoring so as to achieve very high levels of error detection. For critical functions, high levels of reliability are assured by using redundant processor-pairs in duplex or triplex mode. We analyze the reliability of the critical functions of the mission avionics subsystem of the ASID system.

There are several critical functions within the mission avionics system (MAS). The loss of any of these functions causes the system to fail. These critical functions include the vehicle management system (VMS), the crew station control and display functions, mission and systems management, local path generation, and scene and obstacle following functions. The vehicle management system provides airframe control, including flight and propulsion control, as well as providing utility systems management and control. The crew station subsystem displays information to the pilot, contains mechanisms for pilot control actions, and manages crew station activity. The mission and systems management subsystem allocates resources for real time control functions.

Figure 25 is a block diagram of the architecture of the critical mission avionics system. One processing unit is required for the crew station functions, local path generation, and mission and system management. Each of these processing units is supplied with a hot spare backup to take over control if the primary processor should detect an error. Each of the processing units is really a pair of tightly coupled processors so as to maximize the probability of fault detection and minimize latency. Although there are really 4 active processors for each of these functions, we treat the processor-pairs as a single processing unit, since they are not used independently. When a mismatch of results is detected, both members of the processing pair are removed from the system. Figure 25 thus shows that there are two processing units for these functions, where one is the primary unit and the other is a hot spare.

The scene and obstacle and VMS subsystems both require more functionality than one processing unit can provide, and thus each use 2 processing units. The scene and obstacle processing units are also replicated, providing a hot spare backup. The VMS system is triplicated, providing 2 hot spare backups.

In addition to the hot spare backups, 2 additional pools of spares are provided, each containing 2 spare processing units. The first pool can be used to cover the first 2 processor failures in the subsystems other than the VMS; the second pool covers the first 2 failures in the VMS subsystem.

The subsystems are connected via 2 triplicated bus systems, the first being a data bus and the second being the mission management bus. The replicated memory system is connected to the data bus. The VMS has an additional triplicated bus, the vehicle management bus.

7.2 Failure criteria and parameters

The system fails if any of the functions cannot be performed, or if both of the 2 memories fail, or if all 3 of any one type of bus fail. The following MTBF (mean time between failures) values, giving rise to the following failure rates, were used.

- processor pairs: 40,000 hours; failure rate: 2.5×10^{-5}
- buses: 400,000 hours; failure rate: 2.5×10^{-6}
- memories: 1,000,000 hours; failure rate: 1.0×10^{-6}

7.3 Fault recovery

Fault detection is perfect (because of the processing pairs) but it takes between 0.5 second and 5 seconds (uniformly distributed) for recovery to occur. If a second, near-coincident failure occurs during this interval, we say that the system fails because of *near-coincident failures (NCF)*.

7.4 Fault tree model

The fault tree model of the mission avionics system is complicated by the presence of the pooled spares. For ease of exposition, we first present a fault tree model that ignores the pooled spares. We then describe the methodology for modeling pooled spares via a fault tree with sequence dependency gates, by way of a simple example. Finally, we define the full fault tree model of the mission avionics subsystem including the pooled spares.

7.4.1 Fault tree with no pooled spares

The fault tree model of the mission avionics subsystem with no pooled spares is shown in figure 26. This fault tree shows that the system fails if any of the critical functions fail, or if either of the bus systems fail, or if both memories fail. There are 3 types of components in the example system, processing units (type 1), buses (type 2) and memories (type 3). The crew station, for example, uses 2 components of type 1, so its basic event is labeled $2 * 1$. The memory system uses 2 memories and is thus labeled $2 * 3$, while the mission management bus system uses three buses and is labeled $3 * 2$.

7.4.2 Modeling pooled spares

Before we add the pooled spares to the fault tree model of the mission avionics system, consider a simple system with two duplexes and 2 pooled spares. The fault tree model of a 2 duplex system is shown in figure 27, while the equivalent Markov chain is shown in figure 28. This equivalent Markov chain is determined automatically by HARP.

Next, consider the desired Markov chain representation of the same 2-duplex system with the addition of 2 pooled

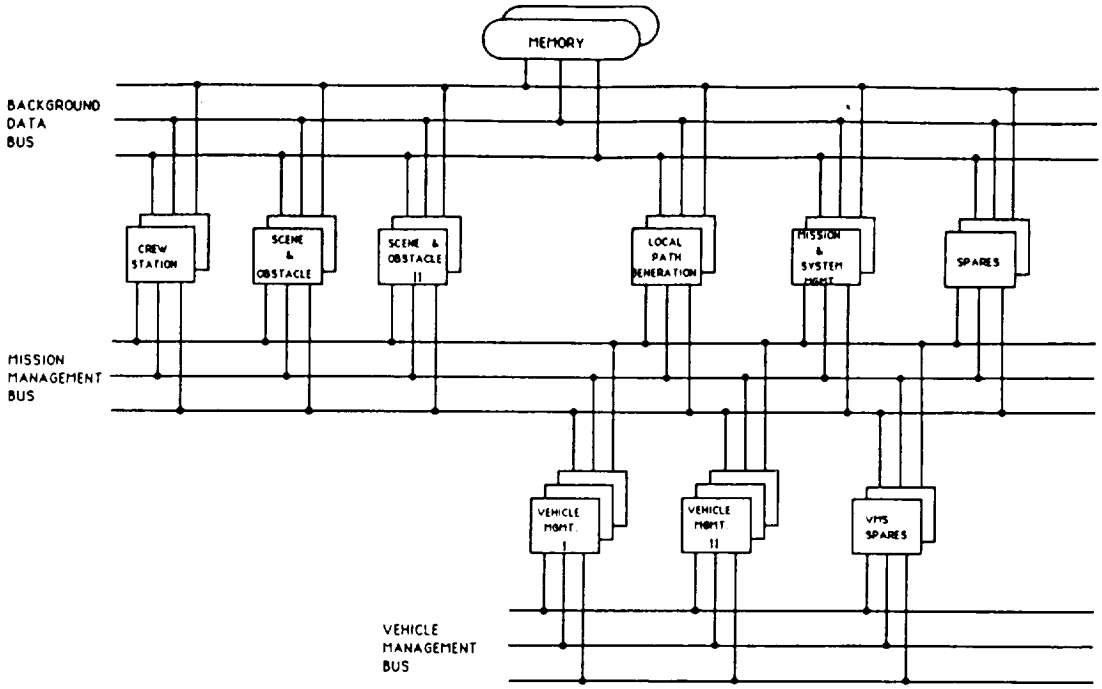


Figure 25: Block diagram of mission avionics system architecture

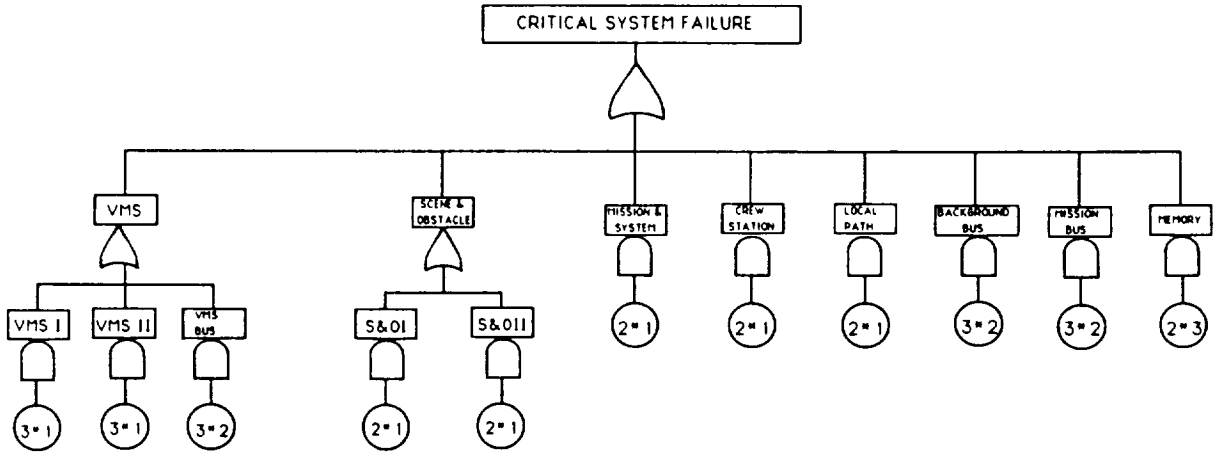


Figure 26: Fault tree model of mission avionics system with no pooled spares

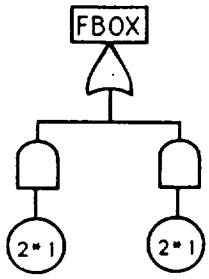


Figure 27: Fault tree model of a 2-duplex system

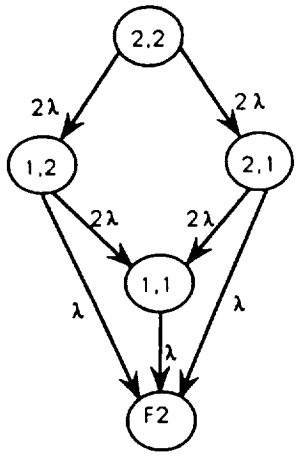


Figure 28: Markov chain model of a 2-duplex system

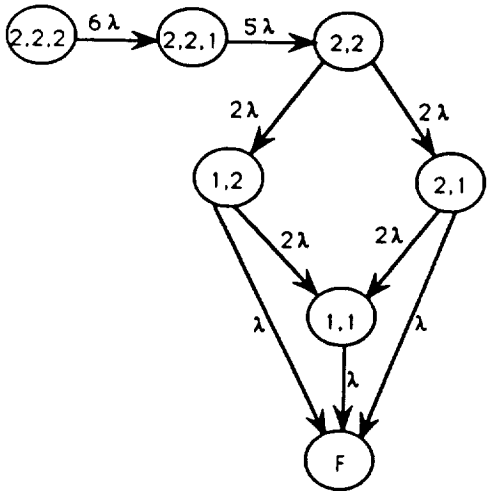


Figure 29: Markov chain model of a 2-duplex system with 2 pooled spares

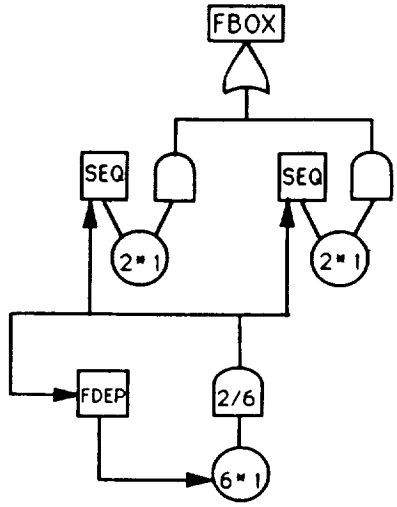


Figure 30: Fault tree model of a 2-duplex system with 2 pooled spares

spares (figure 29). The 2 pooled spares cause 2 states to be added to the front of the Markov chain. These 2 states represent the first 2 failures in the system which will deplete the spares. After the first 2 failures, 2 functioning duplexes remain, and the rest of the Markov chain in figure 29 is identical to that in figure 28.

We can use the fault tree shown in figure 30 to represent the 2-duplex system with 2 pooled spares. In figure 30, the combination of the 2/6 gate (which fires after the first 2 of 6 failures) and the FDEP gate creates a Markov chain that models the first 2 failures of 6 components. After the first 2 failures, the FDEP gate stops any more of the 6 components from failing. The two SEQ gates in figure 30 do not allow the two basic events labeled with $1 * 2$ to begin to fail until after the 2/6 gate has fired. After the 2/6 gate has fired, then the rest of the fault tree (which is identical to the one in figure 27 can occur as usual. This combination of FDEP and SEQ gates can be used in a more general setting to tie multiple Markov chains together.

7.4.3 Full model and results

Figure 31 is the full fault tree model of the mission avionics system, including the pools of spares. The leftmost FDEP and SEQ gates show the 2 spares for the vehicle management system, while those to the right represent the other 2 spares.

Because of the sequence dependency gates, this fault tree cannot be solved by standard combinatorial methods, but rather must be converted to a Markov chain for solution. HARP performs this conversion automatically, and produces a truncated Markov chain with 479 states and 2517 transitions. The Markov model is truncated af-

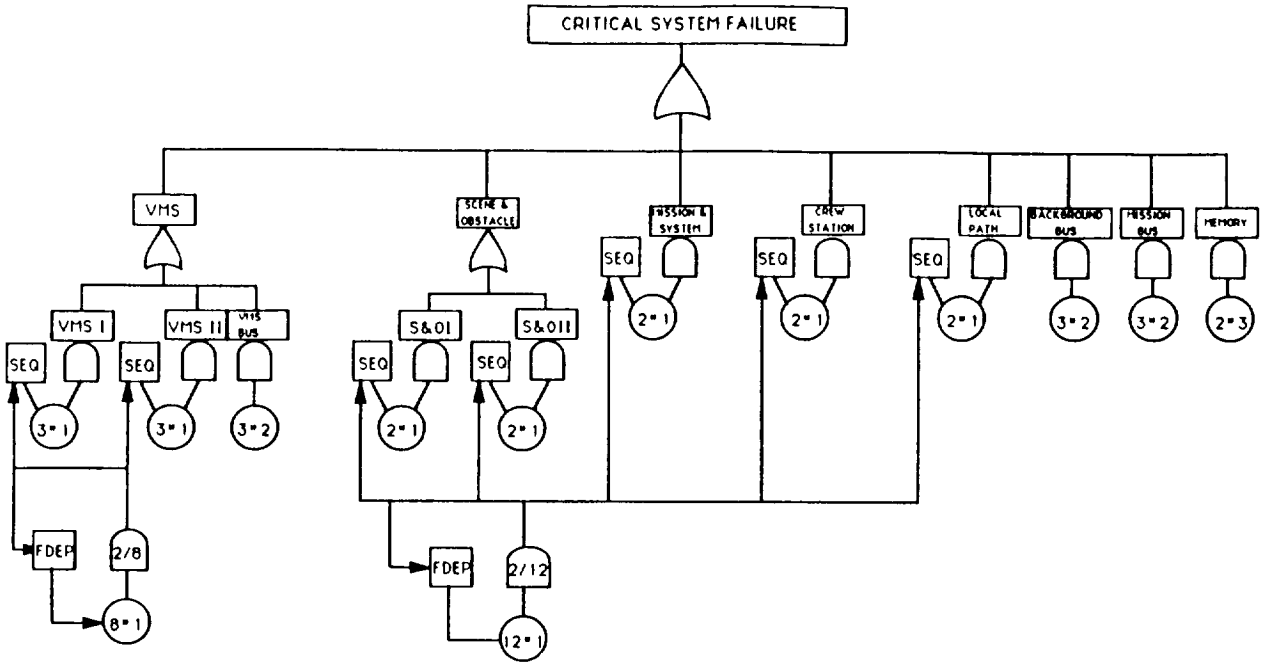


Figure 31: Full fault tree model of mission avionics system

ter considering 5 component failures. Instead of producing an exact reliability estimate, bounds that encompass the reliability of the full model are produced. For a 200 hour interval, the unreliability lies between 1.138×10^{-7} and 1.146×10^{-7}.

8 Three fault tolerant hypercube architectures

We next model three fault tolerant hypercube architectures. All three contain 8 processing nodes connected in a hypercube of dimension 3. All three consist of 2 fault-tolerant modules with each module containing 4 processing nodes. The three architectures differ in the ways that spare nodes are incorporated into the fault-tolerant modules, in the way that messages are routed between processing nodes, and in the architecture of the individual processing nodes. The architectures are described in more detail in a paper that appears in the proceedings of this symposium [5] and are discussed only briefly here.

8.1 System description

8.1.1 Architecture 1

Architecture 1 is based on the hierarchical approach to sparing proposed by Rennels[17] and is depicted in figure 32. It consists of 2 fault tolerant modules of processing nodes. Each module contains 4 processing nodes and one spare node. The spare is connected by a port to each of the 4 active processors in the module.

The processing nodes themselves are comprised of 5 individual processors (4 active processors and a spare) which communicate over an bus and share a memory module. The memory module contains spare bit planes and spare chips within a bit plane. The processing node is connected to its neighboring nodes in the hypercube by 4 ports. Three ports communicate across the three dimensions of the hypercube, and the fourth port communicates with the spare processing node of the module.

8.1.2 Architecture 2

Architecture 2, also depicted in figure 32, is identical to Architecture 1 except that the ports within each processing node are replaced by hyperswitch ports[7]. The hyperswitch allows an adaptive routing method to avoid failed or congested links within the hypercube. It permits any 2 nodes of the hypercube to communicate as long as there exists any nonfailed path between them anywhere throughout the hypercube.

8.1.3 Architecture 3

Architecture 3[6], depicted in figure 33, differs from architectures 1 and 2 in several important ways. Processing nodes are again configured into 2 fault tolerant modules (each containing 4 active processing nodes and one spare), however the inter-node connections are mediated by de-

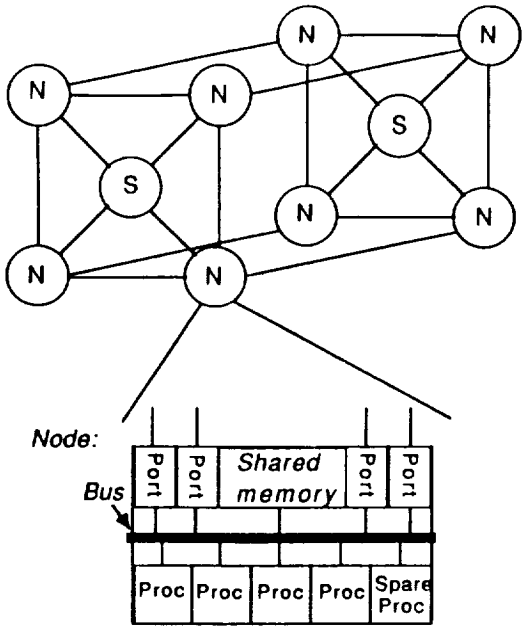


Figure 32: Architecture 1 (Rennels)

coupling switches rather than being direct connections between ports of neighboring nodes. The hypercube connectivity and the switching of spares online and failed nodes offline is performed using these decoupling switches[6]. The switches are intended to be comparatively simple devices. One consequence of using the switches to control access to the spare nodes is that the spares cannot provide redundancy for links as was possible for architectures 1 and 2.

The processing nodes of the hypercube are much simpler and contain processors that are much less powerful that those of architectures 1 and 2. Each processing node consists of 2 processors which perform identical computations in parallel. The output is compared to detect faults. A recovery module is responsible for fault handling upon the detection of a processor fault. The node may either declare itself failed or attempt a reconfiguration to a simplex configuration upon detection of such a processor fault. Both processors have access to a single memory module and a DMA (direct memory access) module. Finally, each processing node communicates with the outside world through three ports, each of which connects the node to its neighbor across one dimension of the hypercube.

For this discussion we examine only the processing nodes of the various candidate architectures in isolation from the ensemble. The processing nodes of each architecture themselves can be configured in a variety of ways. The configuration chosen can affect the reliability and

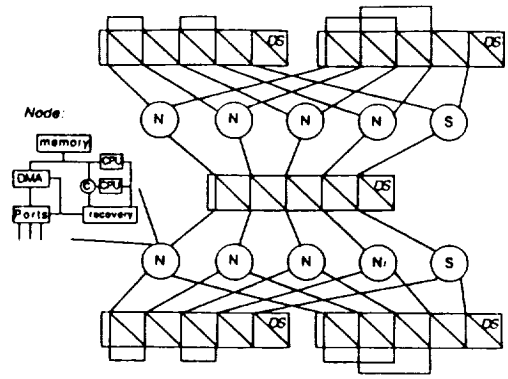


Figure 33: Architecture 3 (Chau and Liestman)

power consumption of the node, which can in turn affect the overall ensemble reliability of the hypercube multiprocessor.

8.2 Failure criteria and parameters

The processor nodes for architectures 1 and 2 are identical, so their failure criteria are closely related. The difference between them is due to the message routing scheme employed by each architecture. A processor node for architectures 1 and 2 will fail if:

- the memory fails OR
- the bus fails OR
- 2 out of the 5 processors fail (the first processor failure is presumably recovered from by switching in the spare to take the failed processor's place) OR
- the node is disconnected from the other processing nodes in the hypercube.

The events that cause a node to be disconnected differ for the two architectures.

The routing algorithm used for architecture 1 allows only one path between each pair of nodes in the hypercube. Since the spare processing node in each of the two fault tolerant modules can relay messages within the module when a direct connection between 2 nodes in the module is not possible, it takes the failure of 2 of the four ports in a processing node to disconnect the node. In Architecture 2, a hyperswitch is used instead of the single path routing algorithm, so that all four ports in a node must fail in order to disconnect the node.

A processing node for architecture 3 fails if:

- the memory fails OR
- the DMA unit fails OR
- both processors fail OR,

- since the single path routing algorithm is used for this architecture, the node will fail if any of its 3 ports fail.

The component failure rates for all three architectures are listed below.

- Active processor (architectures 1 and 2): 1.990×10^{-6} per hour
- Active processor (architecture 3): 2.306×10^{-7} per hour
- Warm spare processor(architecture 2): 1.0×10^{-6} per hour
- Shared Memory (architectures 1 and 2): 3.477×10^{-7} per hour
- Memory (architecture 3): 1.147×10^{-7} per hour
- DMA module (architecture 3): 3.477×10^{-7} per hour
- Intra-node bus(architectures 1 and 2): 1.147×10^{-7} per hour
- Hyperswitch and I/O port (all architectures): 3.477×10^{-7} per hour

8.3 Fault recovery

The FEHM used for the processors assumes that a processor failure can be detected, located, and the spare successfully switched in to replace the failed processor 95% of the time, and that the time required to do all of this is uniformly distributed between 0.9 seconds and 1.1 seconds. The remaining 5% of the time the reconfiguration attempt does not succeed, leading to node failure. The FEHM used for the ports assumes detection and deactivation of a failed port is successful 98% percent of the time, and that the time required for this is exponentially distributed with a mean of 0.1 sec. Again, the remaining 2% of the time a port failure is not successfully detected, leading to node failure. No transient restoration is attempted, i.e., all failures are considered to be permanent.

8.4 Fault tree models

8.4.1 Hot spares

Figures 34 and 35 model the processing nodes in architectures 1 and 2 when the spare processor in the node is a hot spare (the spare is powered on and operating all the time) and hence fails at the same rate as the active processors. The fault trees differ only in the modeling of port failures, as architecture 1 fails when 2 of the four ports fail (hence the 2/4 gate), while architecture 2 doesn't fail until all four ports have failed (hence the AND gate). Figure 36 depicts a fault tree model for the processing nodes of architecture 3.

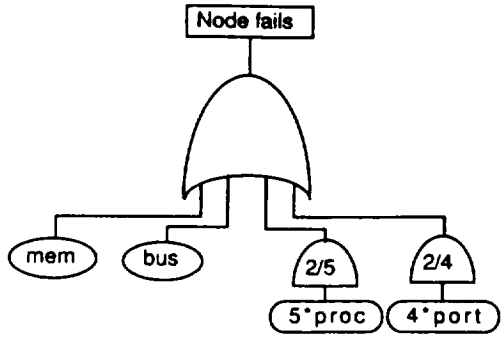


Figure 34: Fault tree model of architecture 1 processing node with hot spares

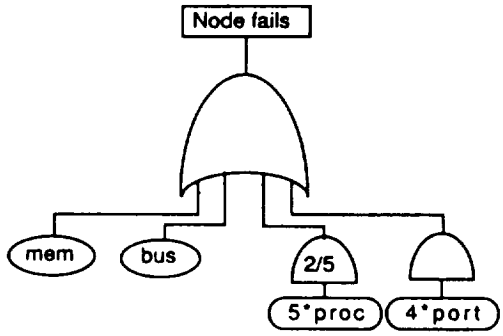


Figure 35: Fault tree model of Architecture 2 processing node with hot spares

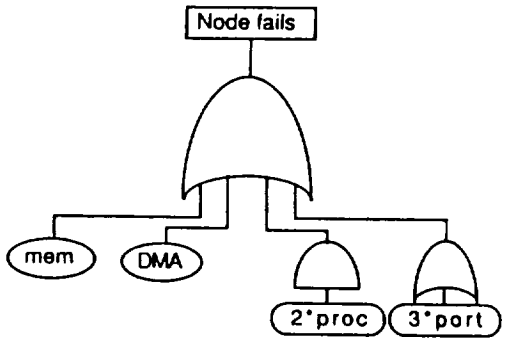


Figure 36: Fault tree model of Architecture 3 processing node with hot spares

8.4.2 Cold spares

Power consumption by a multiprocessor with spare nodes can be reduced by having the spares be *cold spares*, unpowered until they are needed to replace a failed active processor. A cold spare processor cannot fail until it is activated and brought online. In HARP this type of configuration is modeled using the Cold Spare gate, as depicted in figure 37 by a fault tree for architecture 2. The cold spare gate ensures that the spare processor does not fail until one of the 4 active processors fail. The 2/5 gate in parallel with the cold spare gate maintains the requirement that 2 processor failures cause the node to fail. Such a configuration not only reduces power consumption, but also enhances the reliability of the processing node.

8.4.3 Warm spares

Instead of being unpowered, the spare may be partially powered up. It may then fail before being activated but at a lesser rate than the active processors. Such a processor is called a *warm spare* and can be modeled in HARP using the *Sequence Enforcing gate* as shown in figure 38 for architecture 2. In this example two pseudo-components (appearing as inputs to the OR gate whose output feeds into the Functional Dependency gate) are used to represent the 4 active processors and spare before any processor failures. Upon the first failure of a processor (either active or spare), these two pseudo-components are "turned off" as far as the fault tree is concerned by the Functional Dependency gate. The 4 remaining processors, now all active, are represented by the "4*processor" basic event which appears as the rightmost input to the Sequence Dependency gate. This basic event had been "turned off" prior to the first processor failure by the Sequence Enforcing gate. After the first processor failure, the leftmost input to the Sequence Enforcing gate is turned on, which "turns on" the basic event that is its rightmost input (i.e. the processors of this basic event are now permitted to fail). Note that because this basic event is also an input to the top OR gate of the fault tree, a subsequent failure of any of the 4 processors will cause the node to fail, again maintaining the requirement that failure of 2 of the 5 processors cause node failure. Although a spare does not fail while unpowered, upon power up and activation there can be some probability that the spare does not operate properly. Such a situation can be modeled as a warm spare.

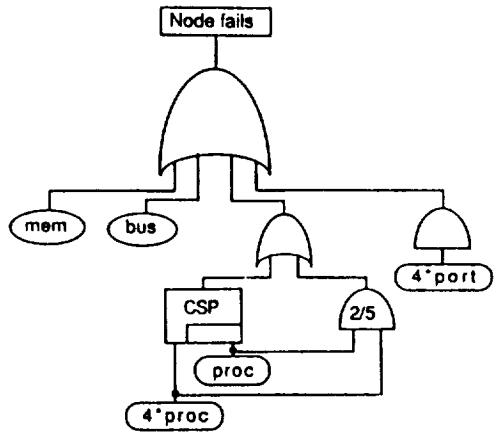


Figure 37: Fault tree model of architecture 2 processing node with cold spares

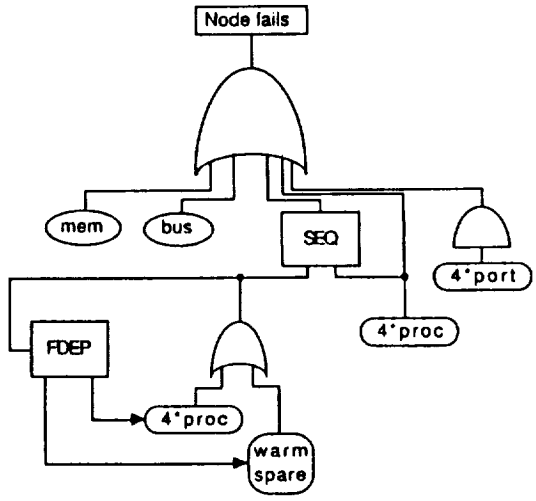


Figure 38: Fault tree model of architecture 2 processing node with warm spares

8.5 Results

Figure 39 compares the 10 year unreliabilities of the processing nodes of each of the three architectures assuming all of them use hot spares. The unreliability of the architecture 3 processing nodes is much lower that those for architectures 1 and 2, reflecting that the reliability of individual processors for architecture 3 is much greater than that of the others and there are only 2 that can fail

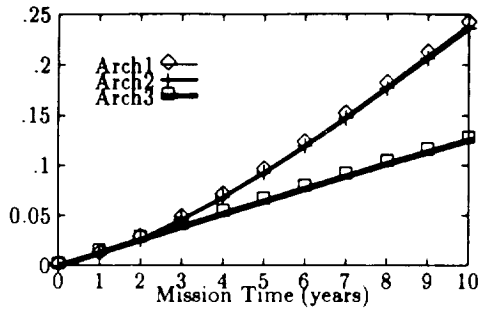


Figure 39: Comparison of node unreliabilities of all three architectures using hot spares

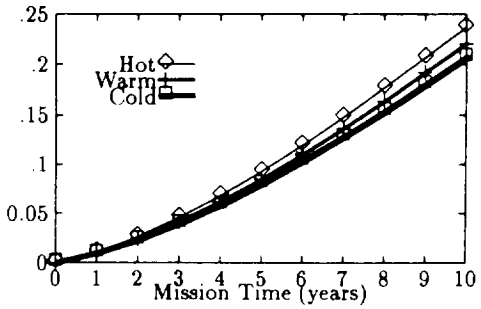


Figure 40: Unreliability of architecture 2 processing nodes with various types of spares

instead of 5. As anticipated, the unreliability for architecture 2 nodes is slightly better than the unreliability for architecture 1 nodes.

Figure 40 shows the 10 year unreliabilities for Architecture 2 processing nodes using hot, warm, and cold spares. In general, the reliability increases from configuration to configuration in that order. This is to be expected, since the failure rate of the spare during its inactive period decreases in that order.

9 References

[1] S. J. Bavuso, J. Bechta Dugan, K. S. Trivedi, E. M. Rothmann, and W. E. Smith. Analysis of typical fault-tolerant architectures using HARP. *IEEE Transactions on Reliability*, R-36(2):176–185, June 1987.

[2] S. J. Bavuso and A. Martensen. A fourth-generation reliability predictor. In *Proceedings of the Reliability and Maintainability Symposium*, pages 11–16, January, 1988.

[3] S. W. Behnen, W. A. Whitehouse, R. J. Farrell, F. M. Leahy, and L. E. Moen. Advanced system integration demonstrations (ASID) system definition. Technical report, AF Wright Aeronautical Laboratories, 1984.

[4] M. A. Boyd, M. Veeraraghavan, J. Bechta Dugan, and K. S. Trivedi. An approach to solving large reliability models. In *AIAA/IEEE Digital Avionics Systems Conference, San Jose, CA*, October 1988.

[5] Mark A. Boyd. Fault tree models for fault tolerant hypercube multiprocessors. In *Proceedings of the Reliability and Maintainability Symposium*, January 1991. Submitted.

[6] S. C. Chau and A. Liestman. Proposal for a fault-tolerant binary hypercube architecture. In *Proc. IEEE Int. Symp. on Fault-Tolerant Computing, FTCS-19*, pages 323–330, June 1989.

[7] E. Chow, J. Peterson, and H. Madan. Hyperswitch network for the hypercube computer. In *Digest of the 13th Symposium on Computer Architecture*, pages 90–99, May 1988.

[8] J. Bechta Dugan, M. Boyd, and S. J. Bavuso. Fault trees and sequence dependencies. In *Proceedings 96th Annual Reliability and Maintainability Symposium*, 1990.

[9] J. Bechta Dugan and K. S. Trivedi. Coverage modeling for dependability analysis of fault-tolerant systems. *IEEE Transactions on Computers*, 38(6):775–787, 1989.

[10] J. Bechta Dugan, K. S. Trivedi, M. K. Smotherman, and R. M. Geist. The hybrid automated reliability predictor. *AIAA Journal of Guidance, Control and Dynamics*, 9(3):319–331, May-June 1986.

[11] J. Bechta Dugan, M. Veeraraghaven, M. Boyd, and N. Mittal. Bounded approximate reliability models for fault tolerant distributed systems. In *Proceedings 8th Symposium on Reliable Distributed Systems*, 1989.

[12] E. Feldman and P. S. Babcock. Reliability evaluation of AIPS I/O networks. Technical Report AIPS-87-15, C. S. Draper Laboratory, Inc., Cambridge, MA, June 1987.

[13] R. E. Harper. Reliability analysis of parallel processing systems. In *Proceedings of the 8th Digital Avionics Systems Conference*, pages 213–219, 1988.

[14] R. E. Harper, J. H. Lala, and J. J. Deyst. Fault tolerant parallel processor architecture overview. In *Proceedings of the 18th Symposium on Fault Tolerant Computing*, pages 252–257, 1988.

[15] E. J. Henley and H. Kumamoto. *Reliability Engineering and Risk Assessment*. Prentice-Hall, 1981.

[16] S. V. Howell and S. J. Bavuso. Fault trees and sequence dependencies. In *Proceedings of the Reliability and Maintainability Symposium*, pages 471–475, January, 1990.

[17] D. A. Rennels. On implementing fault-tolerance in binary hypercubes. In *Proc. IEEE Int. Symp. on Fault-Tolerant Computing, FTCS-16*, pages 344–349, July 1986.

[18] D. P. Siewiorek and R. S. Swarz. *The Theory and Practice of Reliable System Design*. Digital Press, Bedford, MA, 1982.

[19] K. S. Trivedi, R. Geist, M. Smotherman, and J. Bechta Dugan. Hybrid modeling of fault-tolerant systems. *Computers and Electrical Engineering, An International Journal*, 11(2 & 3):87–108, 1985.

References

1. Bavuso, Salvatore J: Hybrid Automated Reliability Predictor Integrated Work Station (HiRel). *Technology 2001: The Second National Technology Transfer Conference and Exposition*, Volume 1, NASA CP-3136, Jan. 1991, pp. 385–394.

2. Bavuso, Salvatore J.; and Dugan, JoAnne B.: HiRel Reliability/Availability Integrated Workstation Tool. *Proceedings of the Annual Reliability and Maintainability Symposium*, IEEE, 1992, pp. 491–500.

3. Platt, M. E.; Lewis, E. E.; and Boehm, F.: *General Monte Carlo Reliability Simulation Code Including Common Mode Failures and HARP Fault/Error-Handling.* NASA CR-187587, 1991.

4. Somani, A. K.; Ritcey, J.; and Au, S.: Computationally-Efficient Phased-Mission Reliability Analysis for Systems With Variable Configurations. *IEEE Trans. Reliab.*, vol. 41, no. 4, Dec. 1992, pp. 504–511.

5. Geist, Robert: Extended Behavioral Decomposition for Estimating Ultrahigh Reliability. *IEEE Trans. Reliab.*, vol. 40, Apr. 1991, pp. 22–28.

6. Dugan, JoAnne Bechta; Trivedi, Kishor S.; Smotherman, Mark K.; and Geist, Robert M.: The Hybrid Automated Reliability Predictor. *AIAA J. Guid., Control & Dyn.*, vol. 9, no. 3, May–June 1986, pp. 319–331.

7. Trivedi, Kishor S.; and Geist, Robert M.: Decomposition in Reliability Analysis of Fault-Tolerant Systems. *IEEE Trans. Reliab.*, vol. 32, no. 5, Dec. 1983, pp. 463–468.

8. Bavuso, S. J.; and Petersen, P. L.: *CARE III Model Overview and User's Guide (First Revision).* NASA TM-86404, 1985.

9. Ng, Ying-Wah; and Avižienis, Algirdas: A Model for Transient and Permanent Fault Recovery in Closed Fault-Tolerant Systems. *Proceedings FTCS-6*, IEEE, 1976, pp. 182–188.

10. Dugan, JoAnne Bechta; Trivedi, Kishor S.; Geist, Robert M.; and Nicola, Victor F.: Extended Stochastic Petri Nets: Applications and Analysis. *Performance '84 Models of Computer System Performance*, E. Gelenbe, ed., Elsevier Sci. Publ. Co., Inc., 1984, pp. 507–519.

11. Geist, Robert; Trivedi, Kishor; Dugan, JoAnne Bechta; and Smotherman, Mark: Design of the Hybrid Automated Reliability Predictor. *Proceedings of the 5th Digital Avionics Systems Conference*, IEEE, 1983, pp. 16.5.1–16.5.8.

12. Trivedi, Kishor; and Dugan, JoAnne Bechta: Hybrid Reliability Modeling of Fault-Tolerant Computer Systems. *Comput. & Electr. Eng.*, vol. 11, no. 2/3, 1984, pp. 87–108.

13. Geist, Robert; Trivedi, K. S.; Dugan, JoAnne Bechta; and Smotherman, Mark: Modeling Imperfect Coverage in Fault Tolerant Systems. *The 14th International Conference on Fault-Tolerant Computing Digest of Papers*, IEEE Comput. Press, 1984, pp. 77–82.

14. Dugan, J. B.; Trivedi, K. S.; Geist, R. M.; and Nicola, V. F.: *Extended Stochastic Petri Nets Applications and Analysis.* AFOSR-84-1095TR, Duke Univ., Nov. 1984. (Available as AD A148 349 from DTIC.)

15. Peterson, James L.: Petri Nets. *Computing Surveys*, vol. 9, no. 3, Sept. 1977, pp. 223–252.

16. Molloy, Michael Karl: On the Integration of Delay and Throughput Measures in Distributed Processing Models. Ph.D. Diss., Univ. of California, 1981.

17. Marsan, Marco Ajmone; and Conte, Gianni: A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Trans. Comput. Sys.*, vol. 2, no. 2, May 1984, pp. 93–122.

18. Trivedi, Kishor Shridharbhai: *Probability and Statistics With Reliability, Queuing, and Computer Science Applications.* Prentice-Hall, 1982.

19. Smotherman, Mark Kelly: Parametric Error Analysis and Coverage Approximations in Reliability Modeling (Sensitivity). Ph.D. Diss., Univ. of North Carolina, 1984.

20. Smotherman, Mark: Error Analysis in Analytic Reliability Modeling. *Microelectron. Reliab.*, vol. 30, no. 1, 1989, pp. 141–149.

21. Smotherman, Mark; Geist, Robert M.; and Trivedi, Kishor S.: Provably Conservative Approximations to Complex Reliability Models. *IEEE Trans. Comput.*, vol. 35, no. 4, Apr. 1986, pp. 333–338.

22. McGough, J.; Smotherman, M.; and Trivedi, K. S.: The Conservativeness of Reliability Estimates Based on Instantaneous Coverage. *IEEE Trans. Reliab. Comput.*, vol. 34, no. 7, July 1985, pp. 602–609.

23. Dugan, JoAnne Bechta; Bavuso, S. J.; and Boyd, M. A.: Modeling Advanced Fault-Tolerant Systems With HARP. Topics in Reliability & Maintainability & Statistics, Consolidated Lecture Notes—1991 "Tutorial Notes," *Proceedings of the Annual Reliability and Maintainability Symposium*, IEEE, 1991, pp. FTS-1–FTS-25.

24. Dugan, J. B.; Bavuso, S. J.: and Boyd, M. S.: Dynamic Fault-Tree Models for Fault-Tolerant Computer Systems. *IEEE Trans. Reliab.*, vol. 41, no. 3, Sept. 1992, pp. 363–376.

25. Dugan, JoAnne Bechta; Bavuso, Salvatore J.; and Boyd, Mark A.: Fault Trees and Sequence Dependencies. *Proceedings of the Annual Reliability and Maintainability Symposium—1990*, IEEE, 1990, pp. 286–293.

26. Dugan, JoAnne Bechta; Bavuso, Salvatore J.; and Boyd, Mark A.: Fault Trees and Markov Models for Reliability Analysis of Fault-Tolerant Digital Systems. *Reliab. Eng. & Sys. Safety*, vol. 39, no. 3, 1993, pp. 291–307.

27. Boyd, Mark Allen: Fault Tree Models—Techniques for Analysis of Advanced Fault Tolerant Computer Systems. Ph.D. Diss., Duke Univ., 1991.

28. Boyd, Mark A.; Veeraraghavan, Malathi; Dugan, JoAnne Bechta; and Trivedi, Kishor S.: An Approach to Solving Large Reliability Models. AIAA-88-3905, 1988.

29. Bavuso, Salvatore J.; Dugan, JoAnne Bechta; Trivedi, Kishor; Rothmann, Beth; and Boyd, Mark: *Applications of the Hybrid Automated Reliability Predictor.* NASA TP-2760, 1987.

30. Bavuso, Salvatore J.; Dugan, JoAnne Bechta; Trivedi, Kishor S.; Rothmann, Elizabeth M.; and Smith, W. Earl: Analysis of Typical Fault-Tolerant Architectures Using HARP. *IEEE Trans. Reliab.*, vol. 36, June 1987, pp. 176–185.

GLOSSARY

Most terms unique to reliability modeling and fault-tolerant systems are defined within the body of each volume of this Technical Paper. The meaning of some terms are well known to researchers and users of these technologies but may not be familiar to new users of Hybrid Automated Reliability Predictor (HARP) integrated reliability tool (HiRel) system. Thus, the purpose of this glossary is to primarily aid new users.

Availability

Availability is a probabilistic quantity that predicts the operational life of a system that is subject to line maintenance (repair). Availability is the probability that a system under repair is operational at a specified time. In a Markov chain model representation, repair is modeled by adding transitions from states with $n+1$ failed components to states with n components. The transition rate is given as a repair rate. No fault tree model representation has yet been developed to represent an availability model; therefore, a Markov chain model must be given to HARP for solution. A fault tree model can be used to specify and generate a preliminary Markov chain model that the user needs to modify.

Behavioral Decomposition

Behavioral decomposition is a mathematical approximation technique that reduces a complex fault/error handling model (FEHM) to a branch point in a Markov chain. The effects of the FEHM are compensated for by modifying state transition rates. The advantage of this technique is that it greatly reduces the size of Markov models for solution and complex FEHM behavior that can be non-Markovian can be modeled.

Bounds or Mathematical Bounds

Large or complex mathematical models often require approximations to keep their solutions tractable. Bounds are the numerical expressions of the variation in a computed result due to mathematical approximation or uncertainty in the accuracy of the input data to the models.

Combinatorial Model

A combinatorial model is a stochastic model that relates combinatorial component failure or success events to a subsystem or system failure or success, respectively. Combinatorial models do not distinguish the order of failure events.

Coincident Fault

A coincident fault exists at the same time one or more other faults are present. A coincident fault is not a simultaneous fault.

Conservative Unreliability Result

Mathematical quantities can be expressed in two forms, in exact form, which is usually a symbolic representation such as the number π, or in an approximate form such as a decimal representation for π as 3.14159. When approximations are necessary, the difference between the exact quantity (which may not be obtainable) and the computed result (which is obtainable) is called the error. A conservative unreliability result is one where the error in the computed result is in the direction of increased unreliability.

Critical-Pair Fault

A critical fault is a near-coincident fault involving two faults. HARP uses three multifault models to account for critical-pair faults: ALL, SAME, and USER.

Extended Behavioral Decomposition

Extended behavioral decomposition is a generalized behavioral decomposition technique that allows multiple FEHM entry/exit transitions and multifault near-coincident modeling.

Fault Tree

A fault tree is a notational model that uses symbols resembling logic gates that relates failure events of components or subsystems to failure events of a system composed of components and subsystems.

Instantaneous Jump Model

An instantaneous jump model is a Markov model that is an approximation of a more complex semi-Markov model that produces a conservative result with respect to the semi-Markov model that is operated on mathematically to become the instantaneous jump model.

Multifault Model

A multifault model is a fault/error handling model that accounts for two or more faults, none occurring simultaneously.

Near-Coincident Fault

A near-coincident fault is second fault that occurs during the time between the occurrence of a first fault and its recovery.

Near-Coincident Failure

A near-coincident failure is system failure resulting from a near-coincident fault. To reduce modeling complexity, a near-coincident failure is assumed to result from a near-coincident fault. Typically, this assumption results in a conservative result.

Optimistic Unreliability Result

An optimistic unreliability result occurs when the error in the computed result is in the direction of decreased unreliability.

Primitive

A primitive is any screen image that is an entity that can be manipulated without dissection, for example, a line, a circle, a fault tree gate, etc.

Semi-Markov Models

Semi-Markov models are generalizations of Markov models. In particular, semi-Markov models allow generalized state holding time distributions. Semi-Markov models are required

for fault-tolerant system models to account for fault/error handling times that may not be exponential.

Sequence-Dependent Model

A sequence-dependent model is a stochastic model that relates ordered component failure or success events to a subsystem or system failure or success, respectively. Sequence-dependent models distinguish the order of failure events. These models are more complex than combinatorial models and are also more difficult to solve.

Simultaneous Fault

A simultaneous fault is a second fault that occurs at exactly the same instant in time as a first fault. Markov chain models do not allow such faults.

Weibull Distribution

A Weibull distribution is a two parameter distribution that can exhibit time increasing, decreasing, or constant failure rates.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1994	Technical Paper

4. TITLE AND SUBTITLE
HiRel: Hybrid Automated Reliability Predictor (HARP) Integrated Reliability Tool System (Version 7.0)
HARP Tutorial

5. FUNDING NUMBERS
WU 505-66-21-02

6. AUTHOR(S)
Elizabeth Rothmann, Joanne Bechta Dugan, Kishor S. Trivedi, Nitin Mittal, and Salvatore J. Bavuso

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-0001

8. PERFORMING ORGANIZATION REPORT NUMBER
L-16553B

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA TP-3452, Vol. 2

11. SUPPLEMENTARY NOTES
Rothmann, Dugan, Trivedi, and Mittal: Duke University, Durham, NC; Bavuso: Langley Research Center, Hampton, VA.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified Unlimited

Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
The Hybrid Automated Reliability Predictor (HARP) integrated Reliability (HiRel) tool system for reliability/availability prediction offers a toolbox of integrated reliability/availability programs that can be used to customize the user's application in a workstation or nonworkstation environment. The Hybrid Automated Reliability Predictor (HARP) tutorial provides insight into HARP modeling techniques and the interactive textual prompting input language via a step-by-step explanation and demonstration of HARP's fault occurrence/repair model and the fault/error handling models. Example applications are worked in their entirety and the HARP tabular output data are presented for each. Simple models are presented at first with each succeeding example demonstrating greater modeling power and complexity. This document is not intended to present the theoretical and mathematical basis for HARP.

14. SUBJECT TERMS
Reliability; Availability; Fault tree; Markov chain; Coverage; Faults; Errors; Fault tolerant; Graphical user interface (GUI)

15. NUMBER OF PAGES
130

16. PRICE CODE
A07

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	